18-661 Introduction to Machine Learning

Course Review and Summary

Spring 2019

Prof. Gauri Joshi
Course Logistics

- **Homework 7**: Jupyter Notebooks due on 28th April.
  - 18-661: Final score will consider the best 3 notebooks
  - 18-461: Final score will consider the best 2 notebooks

- **Practice Exam**:
  - Recitation tomorrow
  - Also in Tuesday’s class

- **Final exam**
  - May 2nd, 6 pm to 9 pm, CIC 4101
  - Two hand-written (two-sided) US-letter size cheat-sheets are allowed
  - Subjective questions, and a few multiple-choice
  - > 75% questions on post-midterm topics
  - Please let us know if you need special accommodations
Goal: Choose the Right ML Method for a Given Task
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Supervised Learning: Parametric Models
Task 1: Regression

How much should you sell your house for?

input: houses & features  learn: $x \rightarrow y$ relationship  predict: $y$ (continuous)

Topics: Feature Scaling, Linear/Ridge Regression, Loss Function, SGD, Regularization, Cross Validation
Linear regression

Setup:

- **Input:** $x \in \mathbb{R}^D$ (covariates, predictors, features, etc)
- **Output:** $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)
- **Model:** $f : x \rightarrow y$, with $f(x) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + w^\top x$.
  - $w = [w_1, w_2, \cdots, w_D]^\top$: weights, parameters, or parameter vector
  - $w_0$ is called bias.
  - Sometimes, we also call $w = [w_0, w_1, w_2, \cdots, w_D]^\top$ parameters.
- **Training data:** $\mathcal{D} = \{(x_n, y_n), n = 1, 2, \ldots, N\}$

Minimize the Residual sum of squares:

$$RSS(w) = \sum_{n=1}^{N} [y_n - f(x_n)]^2 = \sum_{n=1}^{N} [y_n - (w_0 + \sum_{d=1}^{D} w_d x_{nd})]^2$$
RSS(w) in matrix form:

\[
RSS(w) = \sum_{n} [y_n - (w_0 + \sum_{d} w_d x_{nd})]^2 = \sum_{n} [y_n - w^\top x_n]^2,
\]

where we have redefined some variables (by augmenting)

\[
x \leftarrow [1 \; x_1 \; x_2 \; \ldots \; x_D]^\top, \quad w \leftarrow [w_0 \; w_1 \; w_2 \; \ldots \; w_D]^\top
\]

Design matrix and target vector:

\[
X = \begin{pmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_N^\top \end{pmatrix} \in \mathbb{R}^{N \times (D+1)}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \in \mathbb{R}^{N}
\]

Compact expression:

\[
RSS(w) = \|Xw - y\|^2 = \left\{ w^\top X^\top Xw - 2 (X^\top y)^\top w \right\} + \text{const}
\]
Three Optimization Methods

Want to Minimize

\[ RSS(w) = \|Xw - y\|^2_2 = \left\{ w^\top X^\top Xw - 2 (X^\top y)^\top w \right\} + \text{const} \]

- Least-Squares Solution; taking the derivative and setting it to zero

\[ w^{LMS} = (X^\top X)^{-1} X^\top y \]

- Batch Gradient Descent
- Stochastic Gradient Descent
(Batch) Gradient descent

- Initialize $\mathbf{w}$ to $\mathbf{w}^{(0)}$ (e.g., randomly);
  set $t = 0$; choose $\eta > 0$
- Loop until convergence
  1. Compute the gradient
     \[ \nabla \text{RSS}(\mathbf{w}) = \mathbf{X}^\top \mathbf{X} \mathbf{w}^{(t)} - \mathbf{X}^\top \mathbf{y} \]
  2. Update the parameters
     \[ \mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla \text{RSS}(\mathbf{w}) \]
  3. $t \leftarrow t + 1$

What is the complexity of each iteration?
$O(ND)$
Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\mathbf{w}$ to some $\mathbf{w}^{(0)}$; set $t = 0$; choose $\eta > 0$
- Loop until convergence
  1. Randomly choose a training sample $\mathbf{x}_t$
  2. Compute its contribution to the gradient
     $$\mathbf{g}_t = (\mathbf{x}_t^\top \mathbf{w}^{(t)} - y_t)\mathbf{x}_t$$
  3. Update the parameters
     $$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{g}_t$$
  4. $t \leftarrow t + 1$

How does the complexity per iteration compare with gradient descent?

- $O(ND)$ for gradient descent versus $O(D)$ for SGD
What if $X^\top X$ is not invertible?

\[ w^{LMS} = (X^\top X)^{-1} X^\top y \]

Why might this happen?

- **Answer 1:** $N < D$. Not enough data to estimate all parameters.
- **Answer 2:** Columns of $X$ are not linearly independent, e.g., some features are linear functions of other features. In this case, solution is not unique. Examples:
  - A feature is a re-scaled version of another, for example, having two features correspond to length in meters and feet respectively
  - Same feature is repeated twice – could happen when there are many features
  - A feature has the same value for all data points
  - Sum of two features is equal to a third feature
Regularized least square (ridge regression)

**Solution**

\[ w = \left( X^\top X + \lambda I \right)^{-1} X^\top y \]

This is equivalent to adding an extra term to \( \text{RSS}(w) \)

\[
\begin{align*}
\text{RSS}(w) &= \frac{1}{2} \left\{ w^\top X^\top X w - 2 \left( X^\top y \right)^\top w \right\} + \frac{1}{2} \lambda \| w \|_2^2 \\
\text{regularization}
\end{align*}
\]

**Benefits**

- Numerically more stable, invertible matrix
- Force \( w \) to be small
We can use a nonlinear mapping:

\[ \phi(x) : x \in \mathbb{R}^D \rightarrow z \in \mathbb{R}^M \]

- \( M \) is dimensionality of new features \( z \) (or \( \phi(x) \))
- \( M \) could be greater than, less than, or equal to \( D \)

We can apply existing learning methods on the transformed data:

- linear methods: prediction is based on \( w^T \phi(x) \)
- other methods: nearest neighbors, decision trees, etc
Regression with nonlinear basis

Residual sum of squares

\[ \sum_{n}[w^T \phi(x_n) - y_n]^2 \]

where \( w \in \mathbb{R}^M \), the same dimensionality as the transformed features \( \phi(x) \).

The LMS solution can be formulated with the new design matrix

\[
\Phi = \begin{pmatrix}
\phi(x_1)^T \\
\phi(x_2)^T \\
\vdots \\
\phi(x_N)^T 
\end{pmatrix} \in \mathbb{R}^{N \times M}, \quad w^{\text{LMS}} = \left(\Phi^T \Phi\right)^{-1} \Phi^T y
\]
Example with regression

Polynomial basis functions

\[ \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^{M} w_m x^m \]

Fitting samples from a sine function:
underfitting since \( f(x) \) is too simple
Adding high-order terms

\[ M = 3 \]

\[ M = 9: \text{ overfitting} \]

More complex features lead to better results on the training data, but potentially worse results on new data, e.g., test data!
Add a term to the objective function.

- Choose the parameters to not just minimize risk, but avoid being too large.

\[
\frac{1}{2} \left\{ w^\top X^\top X w - 2 (X^\top y)^\top w \right\} + \frac{1}{2} \lambda \| w \|^2
\]
Add a term to the objective function.

- Choose the parameters to not just minimize risk, but avoid being too large.

$$
\frac{1}{2} \left\{ w^\top X^\top X w - 2 (X^\top y)^\top w \right\} + \frac{1}{2} \lambda \| w \|^2_2
$$

Advantages
- Forces the magnitude of $w$ to be small
- Tries to find a simple model with few parameters
- Generalizes well to new data points
Regularization methods

Add a term to the objective function.

• Choose the parameters to not just minimize risk, but avoid being too large.

\[ \frac{1}{2} \left\{ w^\top X^\top X w - 2 (X^\top y)^\top w \right\} + \frac{1}{2} \lambda \| w \|_2^2 \]

Advantages

• Forces the magnitude of \( w \) to be small
• Tries to find a simple model with few parameters
• Generalizes well to new data points
Example: Effect of Regularization

- Regularization makes the higher order $w_i$’s smaller
- Regularized polynomial fit will generalize much better
- As $\lambda$ increases, the model becomes simpler
Tuning $\lambda$ by using a validation dataset

Training data are used to learn $f(\cdot)$. 
N samples/instances: $D^{\text{TRAIN}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$

Test data are used to assess the prediction error.
- M samples/instances: $D^{\text{TEST}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_M, y_M)\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $x \notin D^{\text{TRAIN}}$

Validation data are used to optimize hyperparameter(s).
L samples/instances: $D^{\text{VAL}} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_L, y_L)\}$

Training data, validation and test data should not overlap!
Recipe

• For each possible value of the hyperparameter (say $\lambda = 1, 3, \ldots, 100$)
  • Train a model using $D_{\text{TRAIN}}$
  • Evaluate the performance of the model on $D_{\text{VAL}}$
• Choose the model with the best performance on $D_{\text{VAL}}$
• Evaluate the model on $D_{\text{TEST}}$
Bias-Variance Trade-off: Intuition

- **High Bias:** Model is not rich enough to fit the training dataset and achieve low training loss
- **High Variance:** If the training dataset changes slightly, the model changes a lot
- **Regularization helps find a middle ground**

**Figure 2:** High Bias

**Figure 3:** Just Right

**Figure 4:** High Variance
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks
2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost
3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Task 2: Classification

**Cat or dog?**

**input:** cats and dogs  
**learn:** $x \rightarrow y$ relationship  
**predict:** $y$ (categorical)

**Topics:** Naive Bayes, Logistic Regression, SVMs, Nearest Neighbors, Decision Trees, Random Forests, Boosting, Neural Networks
Intuition: Logistic Regression

- Suppose we want to output the probability of an email being spam/ham instead of just 0 or 1
- This gives information about the confidence in the decision
- Use a function $\sigma(w^T x)$ that maps $w^T x$ to a value between 0 and 1

$\text{SPAM}$
$\text{HAM}$

1
0

$w^T x$, Linear comb. of features

Probability that predicted label is 1 (spam)
• Suppose we want to output the probability of an email being spam/ham instead of just 0 or 1
• This gives information about the confidence in the decision
• Use a function $\sigma(w^T x)$ that maps $w^T x$ to a value between 0 and 1

**Key Problem:** Finding optimal weights $w$ that accurately predict this probability for a new email
Likelihood function

Probability of a single training sample \((x_n, y_n)\)

\[
p(y_n | x_n; w) = \begin{cases} 
\sigma(\mathbf{w}^T \mathbf{x}_n) & \text{if } y_n = 1 \\
1 - \sigma(\mathbf{w}^T \mathbf{x}_n) & \text{otherwise}
\end{cases}
\]

Compact expression, exploring that \(y_n\) is either 1 or 0

\[
p(y_n | x_n; w) = \sigma(\mathbf{w}^T \mathbf{x}_n)^{y_n} [1 - \sigma(\mathbf{w}^T \mathbf{x}_n)]^{1-y_n}
\]
Log Likelihood or Cross Entropy Error

Log-likelihood of the whole training data $\mathcal{D}$

$$
\log P(\mathcal{D}) = \sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log [1 - \sigma(w^\top x_n)]\}
$$

It is convenient to work with its negation, which is called

cross-entropy error function

$$
\mathcal{E}(w) = -\sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log [1 - \sigma(w^\top x_n)]\}
$$

Minimizing via SGD or batch GD

$$
\frac{\partial \mathcal{E}(w)}{\partial w} = \sum_n \{\sigma(w^\top x_n) - y_n\} x_n
$$
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Intuition: Where to put the decision boundary?

\[ \mathbf{w} \cdot \mathbf{x} + b = 0 \]

Idea: Find a decision boundary in the ‘middle’ of the two classes that:

- Perfectly classifies the training data
- Is as far away from every training point as possible
Intuition: Where to put the decision boundary?

Idea: Find a decision boundary in the ‘middle’ of the two classes that:

- Perfectly classifies the training data
- Is as far away from every training point as possible

Let us apply this intuition to build a classifier that MAXIMIZES THE MARGIN between training points and the decision boundary.
Assuming separable training data, we thus want to solve:

\[
\max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|_2} \quad \text{such that} \quad y_n[\mathbf{w}^\top \mathbf{x}_n + b] \geq 1, \quad \forall \ n
\]

This is equivalent to

\[
\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|_2^2
\]

s.t. \( y_n[\mathbf{w}^\top \mathbf{x}_n + b] \geq 1, \quad \forall \ n \)

Given our geometric intuition, SVM is called a max margin (or large margin) classifier. The constraints are called large margin constraints.
We do not want $\xi_n$ to grow too large, and we can control their size by incorporating them into our optimization problem:

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2_2 + C \sum_n \xi_n$$

s.t. $y_n[w^\top x_n + b] \geq 1 - \xi_n, \ \forall \ n$

$\xi_n \geq 0, \ \forall \ n$

What is the role of $C$?

- User-defined hyperparameter
- Trades off between the two terms in our objective
- Same idea as the regularization term in ridge regression, i.e., $C = \frac{1}{\lambda}$
How to solve this problem?

\[
\begin{align*}
\min_{w,b,\xi} \quad & \frac{1}{2} \|w\|^2_2 + C \sum_n \xi_n \\
\text{s.t.} \quad & y_n [w^T x_n + b] \geq 1 - \xi_n, \quad \forall \ n \\
& \xi_n \geq 0, \quad \forall \ n
\end{align*}
\]

- This is a **convex quadratic program**: the objective function is quadratic in \(w\) and linear in \(\xi\) and the constraints are linear (inequality) constraints in \(w\), \(b\) and \(\xi_n\).
- We can solve the optimization problem using general-purpose solvers, e.g., Matlab's \texttt{quadprog()} function.
Visualization of how training data points are categorized

$$w^T \phi(x) + b = 0$$
$$w^T \phi(x) + b = 1$$
$$w^T \phi(x) + b = -1$$

$$\mathcal{H} : w^T \phi(x) + b = 0$$

Support vectors are highlighted by the dotted orange lines

Recall the constraints $y_n[ w^T x_n + b ] \geq 1 - \xi_n$. 
Dual formulation of SVM

Dual is also a convex quadratic program

\[ \max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n \phi(x_m)^\top \phi(x_n) \]

s.t. \[ 0 \leq \alpha_n \leq C, \quad \forall \ n \]
\[ \sum_n \alpha_n y_n = 0 \]

- There are \( N \) dual variables \( \alpha_n \), one for each constraint in the primal formulation
- Independent of \( d \): SVM scales better for high-dimensional data.
We replace the inner products \( \phi(x_m) \trans \phi(x_n) \) with a kernel function

\[
\max_{\alpha} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} y_m y_n \alpha_m \alpha_n k(x_m, x_n)
\]

s.t.  \( 0 \leq \alpha_n \leq C, \forall n \)

\[
\sum_n \alpha_n y_n = 0
\]

We can define the \textbf{kernel function} to learn a nonlinear decision surface.

- We do not need to know the exact form of \( \phi(x) \). E.g., if the kernel is the radial basis function \( k(x, y) = \exp \left( -\|x - y\|^2 \right) \), which roughly measures similarity, it’s not obvious what \( \phi(x) \) should be.
- \( k(x, y) \) is a kernel function if it is symmetric and positive-definite.
- This lets us define much \textbf{more flexible nonlinearities}. 

Recovering solution to the primal formulation

We already identified the primal variable: \[ \frac{\partial L}{\partial w} \rightarrow w = \sum_n \alpha_n y_n \phi(x_n) \]

- Prediction only depends on support vectors, i.e., points with \( \alpha_n > 0 \)

When does \( \alpha_n > 0 \)? This means that:

- \textbf{KKT} complementary slackness conditions tell us:
  
  \begin{align}
  (1) & \quad \lambda_n \xi_n = 0 \\
  (2) & \quad \alpha_n \{1 - \xi_n - y_n [w^T \phi(x_n) + b]\} = 0
  \end{align}

- (2) tells us that \( \alpha_n > 0 \) iff \( 1 - \xi_n = y_n [w^T \phi(x_n) + b] \)
  
  - If \( \xi_n = 0 \), then support vector is on the margin
  - Otherwise, \( \xi_n > 0 \) means that the point is an outlier

- Equality from derivative of Lagrangian yields: (3) \( C - \alpha_n - \lambda_n = 0 \)
  
  - If \( \xi_n > 0 \), then (1) and (3) imply that \( \alpha_n = C \). Then if \( 0 < \alpha_n < C \), \( \xi_n = 0 \).
**Visualizing the support vectors**

\[ w^T \phi(x) + b = 0 \]
\[ H : w^T \phi(x) + b = 1 \]
\[ w^T \phi(x) + b = -1 \]

Support vectors \((\alpha_n > 0)\) are highlighted by the dotted orange lines.

- \(\xi_n = 0\) and \(0 < \alpha_n < C\) when \(y_n[w^T \phi(x_n) + b] = 1\).
- \(\xi_n > 0\) and \(\alpha_n = C\) if \(y_n[w^T \phi(x_n) + b] < 1\).

Recall that the hinge loss is positive whenever \(y_n[w^T \phi(x_n) + b] < 1\), even if the point is correctly classified \((0 < \xi_n < 1)\).
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks
2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost
3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Perceptron: Rosenblatt (1957)

- The perception is a single-unit neural network with the heavyside activation function or \( \text{sign}(x) \)
Perceptron: Rosenblatt (1957)

- The perception is a single-unit neural network with the heavyside activation function or $\text{sign}(x)$.
- It considers a linear binary classification problem to distinguish between two classes $\{-1, +1\}$.

$$g(w^T x + b)$$

Perceptron

$$w^T x + b, \text{ Linear comb. of features}$$
Perceptron: Rosenblatt (1957)

- The perception is a single-unit neural network with the heavyside activation function or \( \text{sign}(x) \).
- It considers a linear binary classification problem to distinguish between two classes \( \{-1, +1\} \).

\[
\begin{align*}
  x_1 & \quad w_1 \\
  x_2 & \quad w_2 \\
  x_3 & \quad w_3 \\
  +1 & \quad b
\end{align*}
\]

\[
g(w^T x + b)
\]

- Assign label \( \text{sign}(w^T x + b) \) to a new sample.
The perception is a single-unit neural network with the heavyside activation function or $\text{sign}(x)$.

It considers a linear binary classification problem to distinguish between two classes $\{-1, +1\}$.

Assign label $\text{sign}(\mathbf{w}^T \mathbf{x} + b)$ to a new sample.

Notation change: Merge $b$ into the vector $\mathbf{w}$ and append 1 to the vector $\mathbf{x}$. 
How to learn the weights $w$?

The objective is to learn $w$ that minimizes the number of errors on the training dataset. That is, minimize

$$\varepsilon = \sum_n \mathbb{1}[y_n \neq \text{sign}(w^\top x_n)]$$

**Algorithm**: For a randomly chosen data point $(x_n, y_n)$ make small changes to $w$ so that

$$y_n = \text{sign}(w^\top x_n)$$
The objective is to learn $\mathbf{w}$ that minimizes the number of errors on the training dataset. That is, minimize

$$
\varepsilon = \sum_n \mathbb{I}[y_n \neq \text{sign}(\mathbf{w}^\top \mathbf{x}_n)]
$$

**Algorithm:** For a randomly chosen data point $(\mathbf{x}_n, y_n)$ make small changes to $\mathbf{w}$ so that

$$
y_n = \text{sign}(\mathbf{w}^\top \mathbf{x}_n)
$$

**Two cases**

- If $y_n = \text{sign}(\mathbf{w}^\top \mathbf{x}_n)$, do nothing.
- If $y_n \neq \text{sign}(\mathbf{w}^\top \mathbf{x}_n)$,

$$
\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w}^{\text{OLD}} + y_n \mathbf{x}_n
$$
Binary Logistic Regression

- Suppose $g$ is the sigmoid function $\sigma(w^T x + b) = \frac{1}{1+e^{-(w^T x + b)}}$
- We can find a linear decision boundary separating two classes. The output is the probability of $x$ belonging to class 1.

Neuron with Sigmoid activation
- Suppose $g$ is the sigmoid function $\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$

- We can find a linear decision boundary separating two classes. The output is the probability of $x$ belonging to class 1.

- This is binary logistic regression, which we already know.
Learning weights $w$ using SGD

Cross-entropy Error Function

$$\mathcal{E}(w) = - \sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)]\}$$
Learning weights $w$ using SGD

**Cross-entropy Error Function**

$$
E(w) = - \sum_{n} \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \}
$$

**Gradients**

$$
\frac{\partial E(w)}{\partial w} = \sum_{n} \{ \sigma(w^\top x_n) - y_n \} x_n
$$

**Gradient descent for logistic regression**

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function

$$
w^{(t+1)} \leftarrow w^{(t)} - \eta \sum_{n} \{ \sigma(w^\top x_n) - y_n \} x_n
$$
Multi-class Regression

- If the target is takes $C$ possible values
- If $y$ belongs to the first class, the outputs should be $[1, 0, \ldots, 0]$
- Need to produce a vector $\hat{y}$ with $\hat{y}_i = p(y = i | x)$
- Linear output layer ($g(x) = x$) first produces un-normalized log probabilities:

$$z = \mathbf{w}^T \mathbf{x} + b$$

Multiclass Regression for $C = 3$
Multi-class Regression

- Softmax:

\[
\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}
\]

- Log of the softmax (since we wish to maximize \(p(y = i; z)\)):

\[
\log \text{softmax}(z)_i = z_i - \log \sum_j \exp(z_j)
\]

Multiclass Regression for \(C = 3\)
Multi-layer Neural Network

- $w_{ij}$: weights connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.

Apply $g_j$ to $u_j$

Nodes in the hidden layer

$y_j = g_j(u_j)$
Multi-layer Neural Network

- \( w_{ij} \): **weights** connecting node \( i \) in layer \( (\ell - 1) \) to node \( j \) in layer \( \ell \).
- \( b_j, b_k \): **bias** for nodes \( j \) and \( k \).
- \( u_j, u_k \): **inputs to nodes** \( j \) **and** \( k \) (where \( u_j = b_j + \sum_i x_i w_{ij} \)).

Apply \( g_j \) to \( u_j \)

Nodes in the hidden layer

\[ y_j = g_j(u_j), \quad z_k = g_k(u_k) \]

\[ t_k \]: target value for node \( k \) in the output layer.
Multi-layer Neural Network

- \( w_{ij} \): **weights** connecting node \( i \) in layer \((\ell - 1)\) to node \( j \) in layer \( \ell \).
- \( b_j, b_k \): **bias** for nodes \( j \) and \( k \).
- \( u_j, u_k \): **inputs to nodes \( j \) and \( k \)** (where \( u_j = b_j + \sum_i x_i w_{ij} \)).
- \( g_j, g_k \): **activation function** for node \( j \) (applied to \( u_j \)) and node \( k \).
- \( y_j = g_j(u_j), z_k = g_k(u_k) \): **output/activation** of nodes \( j \) and \( k \).
Multi-layer Neural Network

- $w_{ij}$: weights connecting node $i$ in layer $(\ell - 1)$ to node $j$ in layer $\ell$.
- $b_j$, $b_k$: bias for nodes $j$ and $k$.
- $u_j$, $u_k$: inputs to nodes $j$ and $k$ (where $u_j = b_j + \sum_i x_i w_{ij}$).
- $g_j$, $g_k$: activation function for node $j$ (applied to $u_j$) and node $k$.
- $y_j = g_j(u_j)$, $z_k = g_k(u_k)$: output/activation of nodes $j$ and $k$.
- $t_k$: target value for node $k$ in the output layer.
SGD Method to Train the neural network weights and biases:

• Step 1: **Forward-propagate to find** $z_k$ in terms of the input (the “feed-forward signals”).

• Step 2: **Calculate output error** $E$ by comparing the predicted output $z_k$ to its true value $t_k$.

• Step 3: **Back-propagate** $E$ by weighting it by the gradients of the associated activation functions and the weights in previous layers.

• Step 4: **Calculate the gradients** $\frac{\partial E}{\partial w}$ and $\frac{\partial E}{\partial b}$ for the parameters $w, b$ at each layer based on the backpropagated error signal and the feedforward signals from the inputs.

• Step 5: **Update the parameters** using the calculated gradients $w \leftarrow w - \eta \frac{\partial E}{\partial w}, b \leftarrow b - \eta \frac{\partial E}{\partial b}$ where $\eta$ is the step size.
Back-propagation Algorithm

Final Layer

- Error in each of its outputs is $z_k - t_k$.
Back-propagation Algorithm

Final Layer

- Error in each of its outputs is $z_k - t_k$.
- Error in input $u_k$ to the final layer is
  $$\delta_k = g'_k(u_k)(z_k - t_k)$$
Back-propagation Algorithm

Final Layer

- Error in each of its outputs is $z_k - t_k$.
- Error in input $u_k$ to the final layer is $\delta_k = g'_k(u_k)(z_k - t_k)$
Final Layer

- Error in each of its outputs is $z_k - t_k$.
- Error in input $u_k$ to the final layer is $\delta_k = g'_k(u_k)(z_k - t_k)$

Hidden Layer

- Error in output $y_j$ is $\sum_{k \in K} \delta_k w_{jk}$.
Final Layer

- Error in each of its outputs is $z_k - t_k$.
- Error in input $u_k$ to the final layer is $\delta_k = g'_k(u_k)(z_k - t_k)$

Hidden Layer

- Error in output $y_j$ is $\sum_{k \in K} \delta_k w_{jk}$.
- Error in the input $u_j$ of the hidden layer is $\delta_j = g'_j(u_j) \sum_{k \in K} \delta_k w_{jk}$
Back-propagation Algorithm

Final Layer

- Error in each of its outputs is \( z_k - t_k \).
- Error in input \( u_k \) to the final layer is \( \delta_k = g_k'(u_k)(z_k - t_k) \)

Hidden Layer

- Error in output \( y_j \) is \( \sum_{k \in K} \delta_k w_{jk} \).
- Error in the input \( u_j \) of the hidden layer is \( \delta_j = g_j'(u_j) \sum_{k \in K} \delta_k w_{jk} \)
Back-propagation Algorithm

**Final Layer**

- Error in each of its outputs is $z_k - t_k$.
- Error in input $u_k$ to the final layer is $\delta_k = g'_k(u_k)(z_k - t_k)$

**Hidden Layer**

- Error in output $y_j$ is $\sum_{k \in K} \delta_k w_{jk}$.
- Error in the input $u_j$ of the hidden layer is $\delta_j = g'_j(u_j) \sum_{k \in K} \delta_k w_{jk}$

The gradient w.r.t. $w_{ij}$ is $x_i \delta_j$. 
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Supervised Learning:
Non-Parametric Models
Nearest neighbor classification (NNC)

Nearest neighbor

\[ x(1) = x_{nn(x)} \]

where \( nn(x) \in [N] = \{1, 2, \cdots, N\} \), i.e., the index to one of the training instances
Nearest neighbor classification (NNC)

Nearest neighbor

\[ x(1) = x_{\text{nn}(x)} \]

where \( \text{nn}(x) \in [N] = \{1, 2, \cdots, N\} \), i.e., the index to one of the training instances

\[ \text{nn}(x) = \arg\min_{n \in [N]} \| x - x_n \|_2^2 = \arg\min_{n \in [N]} \sum_{d=1}^{D} (x_d - x_{nd})^2 \]
Nearest neighbor classification (NNC)

Nearest neighbor

\[ x(1) = x_{\text{nn}(x)} \]

where \( \text{nn}(x) \in [N] = \{1, 2, \cdots, N\} \), i.e., the index to one of the training instances

\[ \text{nn}(x) = \arg\min_{n \in [N]} \| x - x_n \|_2^2 = \arg\min_{n \in [N]} \sum_{d=1}^{D} (x_d - x_{nd})^2 \]

Classification rule

\[ y = f(x) = y_{\text{nn}(x)} \]
In this 2-dimensional example, the nearest point to $x$ is a red training instance, thus, $x$ will be labeled as red.
For every point in the space, we can determine its label using the NNC rule. This gives rise to a decision boundary that partitions the space into different regions.

Increase the number of nearest neighbors that we use?
K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: \( n_{n1}(x) = \arg\min_{n \in [N]} \| x - x_n \|_2^2 \)
- 2nd-nearest neighbor: \( n_{n2}(x) = \arg\min_{n \in [N] - n_{n1}(x)} \| x - x_n \|_2^2 \)
- 3rd-nearest neighbor: \( n_{n2}(x) = \arg\min_{n \in [N] - n_{n1}(x) - n_{n2}(x)} \| x - x_n \|_2^2 \)
K-nearest neighbor (KNN) classification

Increase the number of nearest neighbors to use?

- 1-nearest neighbor: \( \text{nn}_1(x) = \arg\min_{n \in [N]} \| x - x_n \|^2 \)
- 2nd-nearest neighbor: \( \text{nn}_2(x) = \arg\min_{n \in [N] - \text{nn}_1(x)} \| x - x_n \|^2 \)
- 3rd-nearest neighbor: \( \text{nn}_2(x) = \arg\min_{n \in [N] - \text{nn}_1(x) - \text{nn}_2(x)} \| x - x_n \|^2 \)

The set of K-nearest neighbors

\[ \text{knn}(x) = \{ \text{nn}_1(x), \text{nn}_2(x), \ldots, \text{nn}_K(x) \} \]

Let \( x(k) = x_{\text{nn}_k(x)} \), then

\[ \| x - x(1) \|^2 \leq \| x - x(2) \|^2 \leq \cdots \leq \| x - x(K) \|^2 \]

Take majority vote among these neighbors
Example

K=1, Label: ??

K=3, Label: ??

K=5, Label: ??
Example

K=1, Label: red

K=3, Label: red

K=5, Label: blue
How to choose an optimal $K$?

When $K$ increases, the decision boundary becomes smooth.
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
     - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Learning a tree model

Three things to learn:

1. The structure of the tree – order of features to split
2. The threshold values (θᵢ) – these can be categorical
3. The values for the leafs (A, B, C, D, E) – labels that we want to assign to that part of the feature space
Three things to learn:

1. The structure of the tree – order of features to split
2. The threshold values ($\theta_i$) – these can be categorical
3. The values for the leaves ($A, B, \ldots$) – labels that we want to assign to that part of the feature space
Which attribute to split first?

- Patron is a better choice – gives more information to help distinguish between the labels
- Intuition: Like playing 20 questions and choosing carefully which question to ask first
- Idea: use information gain to choose which attribute to split
Which attribute to split first?

- Patron is a better choice – gives more information to help distinguish between the labels
- Intuition: Like playing 20 questions and choosing carefully which question to ask first
- Idea: use information gain to choose which attribute to split
Definition (Conditional Entropy)

Given two random variables $X$ and $Y$

$$H[Y|X] = \sum_k P(X = a_k)H[Y|X = a_k]$$  \hspace{1cm} (1)

In our restaurant example

- $X$: the attribute to be split
- $Y$: Wait or not

Information gain is defined as

$$I(X; Y) = H[Y] - H[Y|X]$$  \hspace{1cm} (2)

When $H[Y]$ is fixed, we need only to compare conditional entropy
Information Gain if we split ”Patron”

- $H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit}$
- $H(Y|X = \text{none}) = 0$
- $H(Y|X = \text{some}) = 0$
- $H(Y|X = \text{full}) = - \left( \frac{2}{2+4} \log \frac{2}{2+4} + \frac{4}{2+4} \log \frac{4}{2+4} \right) \approx 0.9 \text{ bits}$
- Thus the conditional entropy is

$$H(Y|X) = \left( \frac{2}{12} \times 0 + \frac{4}{12} \times 0 + \frac{6}{12} \times 0.9 \right) = 0.45 \text{ bits}$$

- Information Gain $I(X; Y) = 1 - 0.45 = 0.55 \text{ bits}$
Information Gain if we split "Type"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \)
- \( H(Y|X = \text{french}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{italian}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{thai}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{burger}) = \log 2 = 1 \text{ bit} \)
- Thus the conditional entropy is
  \( H(Y|X) = \frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 \text{ bit} \)
- Information Gain \( I(X; Y) = 1 - 1 = 0 \text{ bits} \)
Information Gain if we split "Type"

- \( H(Y) = -\frac{6}{12} \log \frac{6}{12} - \frac{6}{12} \log \frac{6}{12} = 1 \text{ bit} \)
- \( H(Y|X = \text{french}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{italian}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{thai}) = \log 2 = 1 \text{ bit} \)
- \( H(Y|X = \text{burger}) = \log 2 = 1 \text{ bit} \)
- Thus the conditional entropy is
  \( H(Y|X) = \frac{2}{12} \times 1 + \frac{2}{12} \times 1 + \frac{4}{12} \times 1 + \frac{4}{12} \times 1 = 1 \text{ bit} \)
- Information Gain \( I(X; Y) = 1 - 1 = 0 \text{ bits} \)

Thus, we should split on "Patron" and not "Type"
To avoid overfitting a decision tree to a given dataset we can average an ensemble of trees learnt on random subsets of the training data.

**Bagging Trees (Training Phase):**

- For $b = 1, 3, \cdots, B$
  - Choose $n$ training samples $(x_i, y_i)$ from $\mathcal{D}$ uniformly at random
  - Learn a decision tree $h_b$ on these $n$ samples
- Store the $B$ decision trees $h_1, h_2, \ldots, h_B$
- Optimal $B$ (typically in 1000s) chosen using cross-validation

**Bagging Trees (Test Phase):**

- For a test unlabeled example $x$
- Find the decision from each of the $B$ trees
- Assign the majority label as the label for $x$
Random Forests

- **Limitation of Bagging:** If one or more features are very informative, they will be selected by almost every tree in the bag, reducing the diversity.

- **Key Idea on Random Forests:** Reduces correlation between trees in the bag without increasing variance too much.

- Same as bagging in terms of sampling training data.

- Before each split, select $m \leq d$ features at random as candidates for splitting $m \sim \sqrt{d}$.

- Take majority vote of $B$ such trees.
1. Supervised Learning: Parametric Models
   
   Linear Regression
   
   Logistic Regression and Naive Bayes
   
   Support Vector Machines
   
   Neural Networks

2. Supervised Learning: Non-Parametric Models
   
   Nearest Neighbors
   
   Decision Trees and Boosting
   
   AdaBoost

3. Unsupervised Learning
   
   $K$-means clustering
   
   Gaussian mixture models
   
   Principal Component Analysis (PCA)
Limitations of Bagging and Random Forests

- Bagging: Significant correlation between trees that are learnt on different training datasets
- Random Forests try to resolve this by doing “feature bagging” but some correlation still remains
- All $B$ trees are given the same weight when taking the average

**Boosting methods:** Force classifiers to learn on different parts of the feature space, and take their weighted average
**Boosting**

**High-level idea:** combine a lot of classifiers

- Sequentially construct / identify these classifiers, $h_t(\cdot)$, one at a time
- Use *weak* classifiers to arrive at a complex decision boundary (*strong* classifier), where $\beta_t$ is the contribution of each weak classifier

$$h[x] = \text{sign} \left[ \sum_{t=1}^{T} \beta_t h_t(x) \right]$$

**Adaboost Algorithm: Assumptions**

- Black-box routine for constructing a sequence of weak classifiers $h(\cdot)$
- The classifier needs to know how to optimize on *weighted* samples
The Adaboost Algorithm

• Given: $N$ samples $\{x_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some way of constructing weak (or base) classifiers
• Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample
• For $t = 1$ to $T$
  1. Train a weak classifier $h_t(x)$ using current weights $w_t(n)$, by minimizing

$$
\epsilon_t = \sum_n w_t(n)[y_n \neq h_t(x_n)]
$$
The Adaboost Algorithm

- Given: \( N \) samples \( \{x_n, y_n\} \), where \( y_n \in \{+1, -1\} \), and some way of constructing weak (or base) classifiers
- Initialize weights \( w_1(n) = \frac{1}{N} \) for every training sample
- For \( t = 1 \) to \( T \)
  1. Train a weak classifier \( h_t(x) \) using current weights \( w_t(n) \), by minimizing
     \[
     \epsilon_t = \sum_n w_t(n) [y_n \neq h_t(x_n)] \quad \text{(the weighted classification error)}
     \]
  2. Compute contribution for this classifier: \( \beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \)
  3. Update weights on training points
     \[
     w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_nh_t(x_n)}
     \]
     and normalize them such that \( \sum_n w_{t+1}(n) = 1 \)
- Output the final classifier
  \[
  h[x] = \text{sign} \left[ \sum_{t=1}^T \beta_t h_t(x) \right]
  \]
10 data points and 2 features

- The data points are clearly not linearly separable
- In the beginning, all data points have equal weights (the size of the data markers “+” or “-”)
- Base classifier $h(\cdot)$: horizontal or vertical lines (’decision stumps’)
  - Depth-1 decision trees, i.e., classify data based on a single attribute.
Round 1: $t = 1$

- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Weights recomputed; the 3 misclassified data points receive larger weights
Round 2: \( t = 2 \)

- 3 misclassified (with circles): \( \epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65 \).
  Note that \( \epsilon_2 \neq 0.3 \) as those 3 data points have weights less than \( \frac{1}{10} \).
- 3 misclassified data points get larger weights.
- Data points classified correctly in both rounds have small weights.
Round 3: $t = 3$

- $h_3$ misclassified (with circles): $\epsilon = 0.14 \rightarrow \beta = 0.92$.

- Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?

- Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction.
• 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
• Previously correctly classified data points are now misclassified, hence our error is low; what’s the intuition?
Round 3: $t = 3$

- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Previously correctly classified data points are now misclassified, hence our error is low; what’s the intuition?
  - Since they have been consistently classified correctly, this round’s mistake will hopefully not have a huge impact on the overall prediction.
Final classifier: combining 3 classifiers

\[ H_{\text{final}} = \text{sign} \begin{pmatrix} 0.42 \\ + 0.65 \\ + 0.92 \end{pmatrix} \]

- All data points are now classified correctly!
Unsupervised Learning
Task 3: Clustering

How to segment an image?

**input:** raw pixels \( \{x\} \)  
**separate:** \( \{x\} \) into sets  
**output:** cluster labels \( \{z\} \)

**Topics:** K-means clustering, Gaussian Mixture Models, EM
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
**Intuition:** Data points assigned to cluster $k$ should be near prototype $\mu_k$
**Intuition**: Data points assigned to cluster \( k \) should be near prototype \( \mu_k \)

**Distortion measure**: (clustering objective function, cost function)

\[
J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2
\]

where \( r_{nk} \in \{0, 1\} \) is an indicator variable

\[
r_{nk} = 1 \quad \text{if and only if} \quad A(x_n) = k
\]
**k-means Clustering**

**Intuition:** Data points assigned to cluster $k$ should be near prototype $\mu_k$

**Distortion measure:** (clustering objective function, cost function)

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$$

where $r_{nk} \in \{0, 1\}$ is an indicator variable

$$r_{nk} = 1 \quad \text{if and only if} \quad A(x_n) = k$$

**Notes:**

- Distance measure: $||x_n - \mu_k||^2$ calculates how far $x_n$ is from the cluster center $\mu_k$
**Intuition:** Data points assigned to cluster \( k \) should be near prototype \( \mu_k \)

**Distortion measure:** (clustering objective function, cost function)

\[
J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \| x_n - \mu_k \|^2
\]

where \( r_{nk} \in \{0, 1\} \) is an indicator variable

\[
r_{nk} = 1 \quad \text{if and only if} \quad A(x_n) = k
\]

**Notes:**

- Distance measure: \( \| x_n - \mu_k \|^2 \) calculates how far \( x_n \) is from the cluster center \( \mu_k \)
- Canonical example is the 2-norm, i.e., \( \| \cdot \|_2^2 \), but could be something else!
Minimize distortion  Alternative optimization between \( \{r_{nk}\} \) and \( \{\mu_k\} \)

- **Step 0** Initialize \( \{\mu_k\} \) to some values
- **Step 1** Fix \( \{\mu_k\} \) and minimize over \( \{r_{nk}\} \), to get this assignment:
  \[
  r_{nk} = \begin{cases} 
    1 & \text{if } k = \arg\min_j \|x_n - \mu_j\|^2 \\
    0 & \text{otherwise}
  \end{cases}
  \]
- **Step 2** Fix \( \{r_{nk}\} \) and minimize over \( \{\mu_k\} \) to get this update:
  \[
  \mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}
  \]
- **Step 3** Return to Step 1 unless stopping criterion is met
**Initialize**: Pick $k$ random points as cluster centers
k-means example

- **Initialize**: Pick \( k \) random points as cluster centers
- (Shown here for \( k=2 \))
• **Alternating Step 1**: Assign data points to closest cluster center
**Alternating Step 2**: Change the cluster center to the average of the assigned points.

Then: **Repeat …**
$k$-means example (several iterations)
$k$-means example (several iterations)
Elbow method

Key idea: select a small value of $k$ that minimizes within-cluster distances
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
One potential issue with $k$-means . . .

Data points are assigned *deterministically* to one (and only one) cluster.

In reality, clusters may overlap, and it may be better to identify the *probability* that a point belongs to each cluster.
**Key idea:** Model each region with a distinct distribution

- Can use Gaussians — Gaussian mixture models (GMMs)

\[ p(x) = \sum_{k=1}^{K} \omega_k N(x | \mu_k, \Sigma_k) \]

- *However*, we don’t know cluster assignments (label), parameters of Gaussians, or mixture components!

- Must learn from unlabeled data

\[ \mathcal{D} = \{x_n\}_{n=1}^{N} \]
Consider the following joint distribution

\[ p(x, z) = p(z)p(x|z) \]

where \( z \) is a discrete random variable taking values between 1 and \( K \). Denote

\[ \omega_k = p(z = k) \]

Now, assume the conditional distributions are Gaussian distributions

\[ p(x|z = k) = \mathcal{N}(x|\mu_k, \Sigma_k) \]

Then, the marginal distribution of \( x \) is

\[ p(x) = \sum_{k=1}^{K} \omega_k \mathcal{N}(x|\mu_k, \Sigma_k) \]

Namely, the Gaussian mixture model
The complete likelihood is decomposable

\[
\sum_n \log p(x_n, z_n) = \sum_n \log p(z_n) p(x_n | z_n) = \sum_k \sum_{n:z_n=k} \log p(z_n) p(x_n | z_n)
\]

where we have grouped data by cluster labels \(z_n\).

Let \(r_{nk} = \Pr(z_n = k)\), the probability that \(z_n = k\):

\[
\sum_n \log p(x_n, z_n) = \sum_k \sum_n r_{nk} \log p(z = k) p(x_n | z = k)
\]

\[
= \sum_k \sum_n r_{nk} \left[ \log \omega_k + \log N(x_n | \mu_k, \Sigma_k) \right]
\]
Iterative procedure

Alternate between estimating $r_{nk}$ and computing parameters

- Step 0: initialize $\theta$ with some values (random or otherwise)
- Step 1: compute $r_{nk}$ using the current $\theta$
- Step 2: update $\theta$ using the just computed $r_{nk}$
- Step 3: go back to Step 1

This is an example of the **EM algorithm** — a powerful procedure for model estimation with hidden/latent variables
The log-likelihood is

$$\sum_n \log p(x_n, z_n) = \sum_k \sum_n r_{nk} \left[ \log \omega_k + \log N(x_n|\mu_k, \Sigma_k) \right]$$

If $r_{nk}$ was known, we could solve for other parameters as follows:

$$\omega_k = \frac{\sum_n r_{nk}}{\sum_k \sum_n r_{nk}}, \quad \mu_k = \frac{1}{\sum_n r_{nk}} \sum_n r_{nk} x_n$$

$$\Sigma_k = \frac{1}{\sum_n r_{nk}} \sum_n r_{nk} (x_n - \mu_k)(x_n - \mu_k)^\top$$

Since $r_{nk}$ is not given, we estimate it using Bayes’ rule

$$r_{nk} = p(z_n = k|x_n) = \frac{p(x_n|z_n = k)p(z_n = k)}{\sum_{k'}^K p(x_n|z_n = k')p(z_n = k')}$$
Iterative procedure

Alternate between estimating $r_{nk}$ and computing parameters

- Step 0: initialize $\theta$ with some values (random or otherwise)
- Step 1: compute $r_{nk}$ using the current $\theta$
- Step 2: update $\theta$ using the just computed $r_{nk}$
- Step 3: go back to Step 1

This is an example of the EM algorithm — a powerful procedure for model estimation with hidden/latent variables

Connection with $K$-means?

- GMMs provide probabilistic interpretation for $K$-means
- $K$-means is “hard” GMM or GMMs is “soft” $K$-means
- Posterior $r_{nk}$ provides a probabilistic assignment for $x_n$ to cluster $k$
GMMs vs. \textit{k}-means
1. Supervised Learning: Parametric Models
   - Linear Regression
   - Logistic Regression and Naive Bayes
   - Support Vector Machines
   - Neural Networks

2. Supervised Learning: Non-Parametric Models
   - Nearest Neighbors
   - Decision Trees and Boosting
   - AdaBoost

3. Unsupervised Learning
   - $K$-means clustering
   - Gaussian mixture models
   - Principal Component Analysis (PCA)
Task 4: Embedding

How to reduce size of dataset?

Input: large dataset \( \{x\} \)  
Find: sources of variation  
Return: representation \( \{z\} \)

Topics: Dimensionality Reduction, PCA
Issues

1. Measure redundant, high-dimensional signals
2. Represent data via the method by which it was gathered

Goal:
Find a better representation for data

1. To visualize and discover hidden patterns
2. Preprocessing for supervised task

How do we define ’better’?
PCA Formulation

- $X$ is $n \times d$ (raw data)
- $Z = XP$ is $n \times k$ (reduced representation, PCA scores)
- $P$ is $d \times k$ (columns are $k$ principal components)
- Linearity assumption ($Z = XP$) simplifies problem

$$
\begin{bmatrix}
Z \\
\end{bmatrix} = 
\begin{bmatrix}
X \\
\end{bmatrix}
\begin{bmatrix}
P \\
\end{bmatrix}
$$

- Projecting each row of $X$ along the column vectors of $P$
- Question: How to design matrix $P$?
Step 1: Zero-Center All the Features

Given $n$ training points with $d$ features:

- $X \in \mathbb{R}^{n \times d}$: matrix storing points
- $x^{(i)}_j$: $j^{th}$ feature vector for $i^{th}$ point
- $\mu_j$: mean of $j^{th}$ feature
- Deduct $\mu_j$ from all features

\[
\begin{bmatrix}
Z
\end{bmatrix} = \begin{bmatrix}
x_1 - \mu_1 & \cdots & x_d - \mu_d
\end{bmatrix} \begin{bmatrix}
P
\end{bmatrix}
\]

From now on, assume that all columns on $X$ are zero-centered
Eigen-value Decomposition

- The co-variance matrix $\mathbf{C}_X$ is symmetric, positive semi-definite and it can be decomposed as follows

$$
\mathbf{C}_X = \frac{1}{n} \mathbf{X}^T \mathbf{X} \mathbf{Q} \Lambda \mathbf{Q}^T
$$

(5)

$$
\begin{bmatrix}
\mathbf{v}_1 & \cdots & \mathbf{v}_d
\end{bmatrix}
\begin{bmatrix}
\lambda_1 & \mathbf{0} & \cdots & \mathbf{0}
\end{bmatrix}
= \begin{bmatrix}
\mathbf{v}_1^T & \cdots & \mathbf{v}_d^T
\end{bmatrix}
\begin{bmatrix}
\lambda_1 & \mathbf{0} & \cdots & \mathbf{0}
\end{bmatrix}
$$

(6)

- Eigen-values of $\mathbf{C}_X$ are ordered $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$

- $\mathbf{Q}$ is an matrix of $d$ orthonormal eigen-vectors of $\mathbf{C}_X$
• The co-variance matrix $C_X$ is symmetric, positive semi-definite and it can be decomposed as follows

$$C_X = QΛQ^T$$

(7)

$$Q^T C_X Q = Λ$$

(8)

• Recall our desired PCA solution

$$C_Z = \frac{1}{n} Z^T Z$$

$$= \frac{1}{n} P^T X^T X P$$

$$= P^T C_X P$$

• $Z = XP$ is $n \times k$ (reduced representation, PCA scores)

• $C_Z$ should have zero off-diagonal entries
PCA Solution

- The co-variance matrix $C_X$ is symmetric, positive semi-definite and it can be decomposed as follows

$$C_X = Q\Lambda Q^T \quad (9)$$

$$Q^T C_X Q = \Lambda \quad (10)$$

- Choose $P$ as the first $k$ columns of $Q$
- Capture the $k$ directions of maximum variance
- $Z = XP$ is $n \times k$ (reduced representation, PCA scores)
Sparse PCA

PCA’s features are often linear combinations of all the input features. Sparse PCA aims to find sparse linear combinations that also maximize the variance (just like in normal PCA).

If we take $k = 1$ (find the dominant feature):

$$\max_p p^T C_X p$$

$$\text{s.t. } ||p||_2 = 1, \quad ||p||_0 \leq p.$$ (11) (12)

Here $p$, $1 \leq p \leq d$ is a parameter setting the maximum sparsity and $|| \cdot ||_0$ counts the number of nonzero entries in a vector (this is the 0-norm).
We have equipped you with a toolkit of many different ML methods. Choosing the right method is one of the main tasks of a data scientist.

A topic that we did not cover: Reinforcement Learning, more types of neural networks, learning rate schedules.
18-847F: Foundations of Cloud and ML Infrastructure

• Distributed Machine Learning
• Learning Rate Schedules
• Gradient Compression/Quantization
• Hyper-parameter Tuning
• Large-scale Parallel Computing
• Coding-theoretic techniques for Large-scale ML

The format is to read the latest research papers on these topics. Short assignments, no exams, and a final project
Please Complete Course Evaluations!

We will appreciate your constructive feedback on:

- Course Content/Structure
- Level of Math
- TensorFlow assignment
- and my teaching
Course Logistics

- **Homework 7**: Jupyter Notebooks due on 28th April.
  - 18-661: Final score will consider the best 3 notebooks
  - 18-461: Final score will consider the best 2 notebooks

- **Practice Exam**:
  - Recitation tomorrow
  - Also in Tuesday’s class

- **Final exam**
  - May 2nd, 6 pm to 9 pm, CIC 4101
  - Two hand-written (two-sided) US-letter size cheat-sheets are allowed
  - Subjective questions, and a few multiple-choice
  - >75% questions on post-midterm topics
  - Please let us know if you need special accommodations