CS 1675 Introduction to ML
Lecture 3

Introduction to Machine Learning

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square, x4-8845

people.cs.pitt.edu/~milos/courses/cs1675/

Administration

Instructor:
Prof. Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square, x4-8845

TA:
Amin Sobhani
ams543@pitt.edu
6804 Sennott Square
Homework assignment

Homework assignment 1 is out and due on Thursday
Two parts: Report + Programs

Submission:
• via Courseweb
• Report (submit in pdf)
• Programs (submit using the zip or tar archive)
• Deadline 4:00pm (prior to the lecture)

Rules:
• Strict deadline
• No collaboration on the programming and the report part

A learning system: basics

1. Data: \[ D = \{d_1, d_2, \ldots, d_n\} \]
2. Model selection:
   - Select a model or a set of models (with parameters)
     E.g. \[ y = ax + b \]
3. Choose the objective function
   - Squared error \[ \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \]
4. Learning:
   - Find the set of parameters optimizing the error function
     - The model and parameters with the smallest error
5. Testing:
   - Apply the learned model to new data
     - E.g. predict \( y \)s for new inputs \( x \) using learned \( f(x) \)
     - Evaluate on the test data
A learning system: basics

1. Data: \( D = \{d_1, d_2, \ldots, d_n\} \)
2. Model selection:
   - Select a model or a set of models (with parameters)
     E.g. \( y = ax + b \)
3. Choose the objective function
   - Squared error
4. Learning:
   - Find the set of parameters optimizing the error function
     - The model and parameters with the smallest error
5. Testing:
   - Apply the learned model to new data
     - E.g. predict \( y_s \) for new inputs \( x \) using learned \( f(x) \)
     - Evaluate on the test data
A learning system: basics

1. Data: \( D = \{d_1, d_2, \ldots, d_n\} \)

2. Model selection:
   - Select a model or a set of models (with parameters)
     
     E.g. \( y = ax + b \)

3. Choose the objective function
   - Squared error \( \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \)

4. Learning:
   - Find the set of parameters optimizing the error function
     
     The model and parameters with the smallest error

5. Testing:
   - Apply the learned model to new data
     
     E.g. predict \( y \) for new inputs \( x \) using learned \( f(x) \)
   - Evaluate on the test data
A learning system: basics

1. Data: \( D = \{ d_1, d_2, \ldots, d_n \} \)

2. Model selection:
   - Select a model
     - E.g.

3. Choose the objective function
   - Square error

4. Learning:
   - Find the set of parameters optimizing the error function
     - The model and parameters with the smallest error

5. Testing:
   - Apply the learned model to new data
     - E.g. predict \( y_s \) for new inputs \( x \) using learned \( f(x) \)
     - Evaluate on the test data

Testing of learning models

- Simple holdout method
  - Divide the data to the training and test data

- Typically 2/3 training and 1/3 testing
Testing of models

Data set

Training set

Learn on the training set

The model

Evaluate on the test set

Learning process (second look)

1. Data
   – Understand the source of data
   – Real data may need a lot of cleaning/preprocessing
2. Model selection:
   – How to pick the models: manual/automatic methods
3. Choice of the objective (error or loss) function
   – Many functions possible: Squared error, negative log-likelihood, hinge loss
4. Learning:
   – Find the set of parameters optimizing the error function
5. Application/Testing:
   – Evaluate on the test data
   – Apply the learned model to new data
**Data source and data biases**

- **Understand the data source**
- **Understand the data your models will be applied to**
- **Watch out for data biases:**
  - Make sure the data we make conclusions on are the same as data we used in the analysis
  - It is very easy to derive “unexpected” results when data used for analysis and learning are biased

- **Results (conclusions) derived for a biased dataset do not hold in general !!!**

---

**Data**

**Example:** Assume you want to build an ML program for predicting the stock behavior and for choosing your investment strategy

**Data extraction:**
- pick companies that are traded on the stock market on January 2017
- Go back 30 years and extract all the data for these companies
- Use the data to build an ML model supporting your future investments

**Question:**
- Would you trust the model?
- Are there any biases in the data?
Data cleaning and preprocessing

Data may need a lot of:
• Cleaning
• Preprocessing (conversions)

Cleaning:
– Get rid of errors, noise,
– Removal of redundancies

Preprocessing:
– Renaming
– Rescaling (normalization)
– Discretization
– Abstraction
– Aggregation
– New attributes

Data preprocessing

• **Renaming** (relabeling) categorical values to numbers
  – dangerous in conjunction with some learning methods
  – numbers will impose an order that is not warranted
  
  High $\rightarrow$ 2
  Normal $\rightarrow$ 1
  Low $\rightarrow$ 0

  True $\rightarrow$ 2
  False $\rightarrow$ 1
  Unknown $\rightarrow$ 0

  Red $\rightarrow$ 2
  Blue $\rightarrow$ 1
  Green $\rightarrow$ 0

• **Rescaling (normalization):** continuous values transformed to some range, typically [-1, 1] or [0,1].

• **Discretizations (binning):** continuous values to a finite set of discrete values

  ![Discretization Grid](image-url)
Data preprocessing

• **Abstraction:** merge together categorical values

• **Aggregation:** summary or aggregation operations, such as minimum value, maximum value, average etc.

• **New attributes:**
  – example: obesity-factor = weight/height

Model selection

• **What is the right model to learn?**
  – A prior knowledge helps a lot, but still a lot of guessing
  – Initial data analysis and visualization
    • We can make a good guess about the form of the distribution, shape of the function
  – Independences and correlations

• **Overfitting problem**
  – Take into account the **bias and variance** of error estimates
  – Simpler (more biased) model – parameters can be estimated more reliably (smaller variance of estimates)
  – Complex model with many parameters – parameter estimates are less reliable (large variance of the estimate)
Feature selection/dimensionality reduction

Feature/dimensionality reduction selection:
- One way to prevent overfitting for high dimensional data
  \[ x_i = (x_i^1, x_i^2, ..., x_i^d) \quad d \quad \text{very large} \]
- It reduces the dimensionality of data and expresses them in terms of a smaller sets of inputs/features:
  - Feature filtering
  - Multiple features are combined together

Example: document classification
- thousands of documents, >10,000 different words
- Inputs: counts of occurrences of different words
- Overfit threat: too many parameters to learn, not enough samples to justify the estimates the parameters of the model

Solutions for overfitting

How to make the learner avoid overfitting?

- Hold some data out of the training set = validation set
  - Train (fit) on the training set (w/o data held out);
  - Check for the generalization error on the validation set, choose the model based on the validation set error (random re-sampling validation techniques)
Model selection using validation sets

- Select a model from multiple model choices
- Training set is split to training and validation set
- Validation set is used to decide which model is better

![Diagram showing the process of model selection using validation sets.]

Solutions for overfitting

How to make the learner avoid the overfit?

- **Regularization (Occam’s Razor)**
  - Explicit preference towards simple models
  - Penalize for the model complexity (number of parameters) by modifying the objective function

  \[
  \text{Objective function} = \text{error from the data fit} + \text{regularization penalty for the model complexity}
  \]

- Solved through the optimization
Objective criteria

• Measure how well the model fits the data:
  – Mean square error
    \[ w^* = \arg \min_w \text{Error}(w) \quad \text{Error}(w) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i, w))^2 \]
  – Maximum likelihood (ML) criterion
    \[ \Theta^* = \arg \max_\Theta P(D | \Theta) \quad \text{Error}(\Theta) = -\log P(D | \Theta) \]
  – Maximum posterior probability (MAP)
    \[ \Theta^* = \arg \max_\Theta P(\Theta | D) \quad P(\Theta | D) = \frac{P(D | \Theta)P(\Theta)}{P(D)} \]

Other criteria:
  – hinge loss (used in the support vector machines)

Learning

Learning = optimization problem

• Optimization problems can be hard to solve. Right choice of a model and an error function makes a difference.

• Parameter optimizations (continuous space)
  – Linear programming, Convex programming
  – Gradient methods: grad. descent, Conjugate gradient
  – Newton-Rhapson (2nd order method)
  – Levenberg-Marquard
    Some can be carried on-line on a sample by sample basis

• Combinatorial optimizations (over discrete spaces):
  • Hill-climbing
  • Simulated-annealing
  • Genetic algorithms
Parametric optimizations

- Sometimes can be solved directly but this depends on the objective function and the model
  - **Example:** squared error criterion for linear regression
- Very often the error function to be optimized is not that nice.
  \[ Error(w) = f(w) \]
  \[ w = (w_0, w_1, w_2 \ldots w_k) \]
  - a complex function of weights (parameters)
  **Goal:** \[ w^* = \arg \min_w f(w) \]

- **Example of a possible method:** Gradient-descent method
  **Idea:** move the weights (free parameters) gradually in the error decreasing direction

---

Gradient descent method

- Descend to the minimum of the function using the gradient information

\[ Error(w) \]
\[ \frac{\partial}{\partial w} Error(w) |_{w^*} \]

- Change the parameter value of \( w \) according to the gradient
  \[ w \leftarrow w^* + ? \]
Gradient descent method

- Descend to the minimum of the function using the gradient information

\[ Error(w) \]

\[ \frac{\partial}{\partial w} \left. Error(w) \right|_{w^*} \]

- Change the parameter value of w according to the gradient

\[ w \leftarrow w^* - \frac{\partial}{\partial w} \left. Error(w) \right|_{w^*} \]

- New value of the parameter

\[ w \leftarrow w^* - \alpha \frac{\partial}{\partial w} \left. Error(w) \right|_{w^*} \]

\[ \alpha > 0 \quad \text{a learning rate (scales the gradient changes)} \]
Gradient descent method

- To get to the function minimum repeat (iterate) the gradient based update few times

![Gradient descent method diagram](image)

- **Problems**: local optima, saddle points, slow convergence
- More complex optimization techniques use additional information (e.g. second derivatives)

Batch vs on-line learning

- **Batch learning**: Error function looks at all data points
  
  E.g. \( Error(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i, w))^2 \)

- **On-line learning**: separates the contribution from a data point
  
  \( Error_{ON-LINE}(w) = (y_i - f(x_i, w))^2 \)

- **Example**: On-line gradient descent

![Batch vs on-line learning diagram](image)

- **Advantages**: 1. simple learning algorithm
  2. no need to store data (on-line data streams)