Machine Learning
V02: Formulating learning problems

Ingredients to learning
Machine learning from scratch

With material from Andrew Y. Ng, Coursera
Educational objectives

• **Name** the **parts** that make up a machine learning **solution** as well as **concrete instances** of each

• **Understand** the **linear regression** with stochastic **gradient descent** algorithm from scratch

• **Implement** a simple machine **learning algorithm from scratch** (that is, from its mathematical description)
1. INGREDIENTS TO LEARNING
Recap

What is a well-posed learning problem (according to [Mitchell, 1997])?

T: 
P: 
E: 

There are literally thousands of learning algorithms; how can we characterize them?

· 
·
Designing a learning solution

High-level view:

<table>
<thead>
<tr>
<th>Task</th>
<th>Performance measure</th>
<th>training Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.g.,</td>
<td>E.g.,</td>
<td>E.g.,</td>
</tr>
<tr>
<td>• learn playing Tic-Tac-Toe</td>
<td>• #games won against project team</td>
<td>• games played against itself</td>
</tr>
</tbody>
</table>

Practitioner’s view:

Representation: $\mathcal{H}$, features

- E.g.,
  - Full board state as string of X/-/O chars
  - Table mapping strings to winning probability:
    - or neural network?

Evaluation: Loss (objective) function

- E.g.,
  - Sparse rewards: +1 for win, 0 for draw or loss

Optimization: Regime to search $\mathcal{H}$

- E.g.,
  - TD(1) reinforcement learning

Non-trivial: Use prior knowledge / experience on what works under which conditions.

Not so trivial: What features to use? Then pre-processing takes 80% of the time.

Non-trivial: Use prior knowledge / experience on what works under which conditions.

Not so trivial: Often not the standard loss function of your favourite ML library.
Examples for practical components
From [Domingos, 2012]

(Not all possible tuples of $<representation, evaluation, optimization>$ exist / make sense)

<table>
<thead>
<tr>
<th>Representation</th>
<th>Evaluation</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances</td>
<td>Accuracy/Error rate</td>
<td>Combinatorial optimization</td>
</tr>
<tr>
<td>K-nearest neighbor</td>
<td>Precision and recall</td>
<td>Greedy search</td>
</tr>
<tr>
<td>Support vector machines</td>
<td>Squared error</td>
<td>Beam search</td>
</tr>
<tr>
<td>Hyperplanes</td>
<td>Likelihood</td>
<td>Branch-and-bound</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Posterior probability</td>
<td>Continuous optimization</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>Information gain</td>
<td>Unconstrained</td>
</tr>
<tr>
<td>Decision trees</td>
<td>K-L divergence</td>
<td>Gradient descent</td>
</tr>
<tr>
<td>Sets of rules</td>
<td>Cost/Utility</td>
<td>Conjugate gradient</td>
</tr>
<tr>
<td>Propositional rules</td>
<td>Margin</td>
<td>Quasi-Newton methods</td>
</tr>
<tr>
<td>Logic programs</td>
<td></td>
<td>Constrained</td>
</tr>
<tr>
<td>Neural networks</td>
<td></td>
<td>Linear programming</td>
</tr>
<tr>
<td>Graphical models</td>
<td></td>
<td>Quadratic programming</td>
</tr>
<tr>
<td>Bayesian networks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional random fields</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In a nutshell: Use experience (own or read), experiment a lot, constrain solutions

How to select? (we come back to this question often…)

- Remember V01: No generally best solution available (no free lunch) → see V03 and V06 for best practices on model selection
- Guide: «What prior knowledge is easily expressed in certain features & models?»
- Relieve: Good & compact features are more important than model choice

Ask yourself: «Do I see the sought patterns in these features alone?»
Suppose we feed a learning algorithm a lot of historical weather data, and have it **learn to predict weather**. In this setting, **what is** its training experience $E$?

- None of these
- The probability of it correctly predicting a future date’s weather
- The process of the algorithm examining a large amount of historical weather data
- The weather prediction task
Formulating a machine learning solution
Quizzy 2/5

Suppose we are working on weather prediction, and use a learning algorithm to predict tomorrow’s temperature (in degrees Celsius). Would you treat this as a classification or a regression problem?

- Classification
- Regression
Formulating a machine learning solution
Quizzy 3/5

Suppose we are working on stock market prediction, and we would like to predict whether or not a particular stock's price will be higher tomorrow than it is today. You want to use a learning algorithm for this. Would you treat this as a \textbf{classification} or a \textbf{regression} problem?

- Classification

- Regression
Formulating a machine learning solution
Quizzy 4/5

Some of the problems below are best addressed using a supervised learning algorithm, and the others with an unsupervised algorithm. Which of the following would you apply supervised learning to?

- Examine a web page, and classify whether the content on the web page should be considered “child friendly” (e.g., non-pornographic etc.) or ”adult”

- Examine a large collection of emails that are known to be spam email, to discover if there are sub-types of spam mail

- In farming, given data on crop yields over the last 50 years, learn to predict next year’s crop yields

- Take a collection of 1’000 essays written on the Swiss economy, and find a way to automatically group these essay into a small number of groups that are somehow “similar” or “related”
Formulating a machine learning solution

Quizzy 5/5

Many substances that can burn…
…(such as gasoline and alcohol) have a chemical structure based on carbon atoms. For this reason they are called hydrocarbons. A chemist wants to understand how the number of carbon atoms in a molecule affects how much energy is released when that molecule combusted (meaning that it is burned). The chemist obtained the dataset below. In the column on the right, "kJ/mol" is the unit measuring the amount of energy released.

Is it classification or regression?
What is $X$, $Y$ (the training data)?
What could be the relationship? How to gain first insight?

<table>
<thead>
<tr>
<th>Name of molecule</th>
<th>Number of carbon atoms in molecule</th>
<th>Heat released when burned (kJ/mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methane</td>
<td>1</td>
<td>-890</td>
</tr>
<tr>
<td>Ethene</td>
<td>2</td>
<td>-1411</td>
</tr>
<tr>
<td>Ethane</td>
<td>2</td>
<td>-1560</td>
</tr>
<tr>
<td>Propane</td>
<td>3</td>
<td>-2220</td>
</tr>
<tr>
<td>Cyclopropane</td>
<td>3</td>
<td>-2091</td>
</tr>
<tr>
<td>Butane</td>
<td>4</td>
<td>-2878</td>
</tr>
<tr>
<td>Pentane</td>
<td>5</td>
<td>-3537</td>
</tr>
<tr>
<td>Benzene</td>
<td>6</td>
<td>-3268</td>
</tr>
<tr>
<td>Cycloexane</td>
<td>6</td>
<td>-3920</td>
</tr>
<tr>
<td>Hexane</td>
<td>6</td>
<td>-4163</td>
</tr>
<tr>
<td>Octane</td>
<td>8</td>
<td>-5471</td>
</tr>
<tr>
<td>Naphthalene</td>
<td>10</td>
<td>-5157</td>
</tr>
</tbody>
</table>
Getting first insights
Example: Quizzy 5/5

Low-hanging fruits
• Exploratory data analysis (visualization) → this and next slide
• Trying simpler models first → next section

import matplotlib.pyplot as plt
import pandas as pd

data_frame = pd.read_excel("hydrocarbons.xlsx")

plt.scatter(data_frame['nr_molecules'], data_frame['heat_release'])
plt.title("Scatter Plot X vs. Y")
plt.xlabel(data_frame.columns[1])
plt.ylabel(data_frame.columns[2])
plt.show()
The Machine Learning development process
Exploration & experimentation

Necessity of a distinct conceptual approach

- Modeling data ≠ {software dev., business process mgmt., data base design, stat. analysis, …}

In a nutshell

- Focus on **systematic experimentation** and **rigorous evaluation** → automatized
- Best implemented by (a **pipeline of** scripts) → UNIX command line approach
- Data exploration and rapid prototyping is key → IPython /Jupyter (see appendix of V03)
2. MACHINE LEARNING FROM SCRATCH
Carbon and combustion
Continuous example for this section

We need a solid understanding of <Representation, Evaluation, Optimization>

Thus, this section explores
• …a «straight line fit» as a simplistic model
• …all the details of how to train it
• …to get a feeling of ML apart from libraries

Observations
• The data shows a linear trend
• The gradient is roughly \[ \frac{-2500+2600}{2-4} = -50 \]

Conclusions
• Labeled data available \(\rightarrow\) supervised learning
• Continuous valued output \(\rightarrow\) regression
• Reasonable straight line fit \(\rightarrow\) linear regression could be a first try
How to represent $h$? 
..in the case of univariate linear regression 

$$h(x, \theta) = \theta_0 + \theta_1 x$$

• $\theta_0 =$ intercept, $\theta_1 =$ gradient 

How to choose parameters $\theta_0$, $\theta_1$? 
• Parameters correspond with different fits 

Training Set 

Learning Algorithm (optimizer) 

$x$: carbon portion in molecule 

$h$ 

$y$: estimated fuel value
Cost function $J$

Driver of the optimization

Idea
- **Choose $\theta_0$, $\theta_1$** so that $h(x)$ is close to $y$ for the training examples $(x, y)$
- Let a function $J(parameters)$ **number the cost of errors** made for specific parameters

\[
J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left( h(x_i, \theta) - y_i \right)^2
\]

Precise objective
- **Minimize $J$** w.r.t. $\theta_0$, $\theta_1$
- In terms of V01:
  - $L(\hat{y}, y) = (\hat{y} - y)^2$ is the squared error loss function
  - $J(\theta_0, \theta_1, h, X, Y, L) = \text{Emp} \left( h(\theta), X, Y, L \right) = \frac{1}{2N} \sum_{i=1}^{N} L \left( h(x_i, \theta), y_i \right)$ is the **cost of error** with explicit respect to **all parameters**: Measured in terms of the empirical error over the training set $(X, Y)$ according to the squared error loss function $L$ for a particular hypothesis $h$
Simplified version to gain intuition
Fixed intercept at (0,0)

- Hypothesis: \( h(x, \theta_1) = 0 + \theta_1 x \)
- Parameter: \( \theta_1 \)
- Cost: \( J(\theta_1) = \frac{1}{2N} \sum_{i=1}^{N} (h(x_i, \theta_1) - y_i)^2 \)
- Goal: minimize \( J(\theta_1) \)

Example values of \( J \)
- \( J(1) = \frac{1}{6} \sum_{i=1}^{3} (h(x_i, 1) - y_i)^2 = \frac{1}{6} \sum_{i=1}^{3} (x_i - y_i)^2 = \frac{1}{6} ((1-1)^2 + (2-2)^2 + (3-3)^2) = 0 \)
- \( J\left(\frac{1}{2}\right) = \ldots = \frac{1}{6} \left((\frac{1}{2}-1)^2 + (1-2)^2 + (\frac{3}{2}-3)^2\right) = \frac{1}{6} \cdot \frac{7}{2} \approx 0.58 \)
- \( J(0) = \ldots = \frac{1}{6} (1^2 + 2^2 + 3^2) = \frac{1}{6} \cdot 14 \approx 2.33 \)
- \( \ldots \)
Back to two parameters

- Hypothesis: \( h(x, \theta) = \theta_0 + \theta_1 x \)
  - For fixed \( \theta \), this is a function of \( x \)

- Parameters: \( \theta_0, \theta_1 \)

- Cost: \( J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left( h(x_i, \theta) - y_i \right)^2 \)
  - For fixed \( x \), this is a function of \( \theta \)

- Goal: minimize \( J(\theta_0, \theta_1) \)

Example: Housing prices \( (y) \) per size \( (x) \)
- \( h(x, \theta) \)

\[ h(x, \theta) \] is a convex (bowl-shaped) function of 2 variables \( \rightarrow 3D \)

...and its 2D visualization as a contour plot
Back to two parameters contd.

Example: Housing prices \((y)\) per size \((x)\)

- \(h(x, \theta)\)

\[ h(x, \theta) = h\left(x, \theta_0, \theta_1\right) \]

\[ J(\theta_0, \theta_1) \]

\[
\begin{align*}
\hat{h}(x, \theta) &= h(x, 800, -0.15) \\
\hat{h}(x, \theta) &= h(x, 500, 0.025) \\
\hat{h}(x, \theta) &= h(x, 360, 0) \\
\hat{h}(x, \theta) &= h(x, 100, 0.14) \\
\end{align*}
\]
Optimization by gradient descent
Numerical optimization

Have: Some function $J(\theta_0, \theta_1)$
Want: minimize $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Algorithm
• Start with some $\theta_0, \theta_1$ (e.g., (0,0))
• Keep changing $\theta_0, \theta_1$ to reduce $J(\theta_0, \theta_1)$
  → Direction: steepest descent
• End: Hopefully at a minimum

Observations
• Small changes in starting point result in different local minima
• Assumption: cost surface is smooth, local minima are ok
Gradient descent algorithm

Pseudo code for gradient descent

• repeat until convergence:
  for j:=0..1:
    \( \theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0,\theta_1) \)
  for j:=0..1:
    \( \theta_j := \hat{\theta}_j \)

• \( \frac{\partial}{\partial \theta_j} J(\theta_0,\theta_1) \) is the partial derivative of \( J \) w.r.t. \( \theta_j \)

• \( \alpha > 0 \) is called the learning rate (it is a data-dependent hyperparameter of the algorithm)

• Important: simultaneous update!

Why not solving it analytical?

• Numerical optimization scales better to larger data sets

• Gradient descent also works for \( h \)'s without analytical solution (e.g., neural networks)
Intuition behind gradient descent formulae (II)
Effect of the learning rate $\alpha$

- $\alpha$ too small
  $\Rightarrow$ gradient descent is slow

- $\alpha$ too large
  $\Rightarrow$ gradient descent overshoots minimum
  $\Rightarrow$ no convergence or even divergence!

\[ J(\theta) \]
\[ \theta \]
Gradient descent for univariate linear regression

Formal overview

Ingredients

- **Representation**
  - \( h(x, \hat{\theta}) = \theta_0 + \theta_1 x \)

- **Evaluation**
  - \[ J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left( h(x_i, \hat{\theta}) - y_i \right)^2 \]
  - \[ \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \left( h(x_i, \hat{\theta}) - y_i \right) \cdot \frac{\partial}{\partial \theta_0} h(x_i, \hat{\theta}) \]
  - \[ \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \left( h(x_i, \hat{\theta}) - y_i \right) \cdot \frac{\partial}{\partial \theta_1} h(x_i, \hat{\theta}) \]

- **Optimization**
  - repeat until convergence:
    - for \( j := 0 \ldots 1 \):
      - \( \hat{\theta}_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \)
    - for \( j := 0 \ldots 1 \):
      - \( \theta_j := \hat{\theta}_j \)

How to take the derivative:
\( \rightarrow \) see appendix

Batch gradient descent: uses all training examples at once (as opposed to stochastic gradient descent, which uses small chunks called “mini-batches”…)

Repeat until convergence:
- \( \theta_0 := \theta_0 - \alpha \cdot \frac{1}{N} \sum_{i=1}^{N} \left( h(x_i, \hat{\theta}) - y_i \right) \cdot x_i \)
- \( \theta_1 := \theta_1 - \alpha \cdot \frac{1}{N} \sum_{i=1}^{N} \left( h(x_i, \hat{\theta}) - y_i \right) \cdot x_i \)
Review

• A learning solution needs a representation, an evaluation function and an optimizer

• These can be derived from the formulation of a well-posed learning problem as task, performance measure and training experience

• There is no general solution to deriving these concrete methods. It is problem (data-) dependent and relies on prior knowledge

• Valid guides are the characteristics of methods (inductive bias, VC theory), experience / best practices and prior knowledge

• Gradient descent is a general-purpose optimizer; implementation details (simultaneous updates) and hyperparameters are practically very relevant
P02.1: Implementing ML from scratch

Work through exercise P02.1:

• Implement the algorithms derived in this chapter just using the given formal descriptions (i.e., slide 24)

• Reflect on the methods: How transferable are experiences from one data set to the next?

• Reflect on your implementation: What took you the most time? Which part was easy for you?
APPENDIX
Remark: Different levels of inductive bias
Are there more general forms of prior knowledge that universally guide learning?

- There's a linear relationship between inputs & outputs.
- The hypothesis space is smooth.
- Learn sparse, distributed representations.
Intuition behind gradient descent formulae (I)

Signs and $\alpha$ automatically care for proper descent

- $\theta'' := \theta' - \alpha \cdot \frac{\partial}{\partial \theta'} J(\theta')$
- $\theta'' := \theta' - \alpha \cdot \frac{\partial}{\partial \theta'} J(\theta')$

As we approach the minimum, steps automatically get smaller $\Rightarrow \alpha$ may be fixed over time
Derivative of \( J \) w.r.t. \( \theta_j \)

\[
\frac{\partial}{\partial \theta_j} J(\theta_o, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2N} \sum_{i=1}^{N} (h(x_i, \overline{\theta}) - y_i)^2 = \sum_{i=1}^{N} \frac{\partial}{\partial \theta_j} \frac{1}{2N} (h(x_i, \overline{\theta}) - y_i)^2
\]

Chain rule: \( f(g(x))' = f'(g(x)) \cdot g'(x) \)

\[
= \sum_{i=1}^{N} \frac{2}{2N} (h(x_i, \overline{\theta}) - y_i) \cdot \frac{\partial}{\partial \theta_j} (h(x_i, \overline{\theta}) - y_i)
\]

\[
= \sum_{i=1}^{N} \frac{1}{N} (h(x_i, \overline{\theta}) - y_i) \cdot \frac{\partial}{\partial \theta_j} h(x_i, \overline{\theta})
\]

\[
\begin{cases}
  j = 0 & \rightarrow \frac{1}{N} \sum_{i=1}^{N} (h(x_i, \overline{\theta}) - y_i) \cdot 1 \\
  j = 1 & \rightarrow \frac{1}{N} \sum_{i=1}^{N} (h(x_i, \overline{\theta}) - y_i) \cdot x_i
\end{cases}
\]
Choosing cost functions

Ideal properties of a cost function
1. Being **easy to optimize** \(\Rightarrow\) should be a *convex* function
2. Assigning **equal cost to far and very far** off examples \(\Rightarrow\) makes it **robust to outliers**

Cost functions in practice
- MSE (mean-squared error) is almost always used for regression \(\Rightarrow\) it only exhibits property 1
- Making MSE level off would make the function non-convex \(\Rightarrow\) when using **MSE**, one has to care for **outliers** during **pre-processing**
- Cost function design is important
  (because the usual one might not capture the problem well)
- …but care has to be taken to make it mathematically sound!

Further reading
- Boyd & Vandenberghe, «*Convex Optimization*», 2004 \(\rightarrow\) ch. 3
- Bertsekas, «*Convex Optimization Algorithms*», 2015 \(\rightarrow\) ch. 1
- Chu, «*Machine Learning Done Wrong*», 2015
Examples of built-to-purpose cost functions
from [Mitchell, 1997], chapter 6.5

Certain well-known cost functions can be justified theoretically using Bayesian reasoning by showing optimality under certain assumptions:

Minimizing *squared error*

- Yields maximum likelihood (ML) hypothesis assuming Gaussian noise on the labels
  
  Example: Training *linear regression* to fit a straight line

Minimizing *cross entropy*

- Yields ML hypothesis assuming the labels are a probabilistic function of the training examples
  
  Example: Training a *neural network* to predict probabilities

→ see appendix of V03

CMU’s Tom Mitchell, author of one of the most instructive machine learning books.