Introduction to Machine Learning

Linear Classification

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)
Linear classifiers

Data set \( D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \) where \( x \in \mathbb{R}^d \)

\[ h(x) = \text{sign}(\mathbf{w}^T \mathbf{x}) \]

Decision boundary: \( \mathbf{w}^T \mathbf{x} = 0 \)

\[ \{x \mid \mathbf{w}^T \mathbf{x} > 0\} \]

\[ \{x \mid \mathbf{w}^T \mathbf{x} < 0\} \]
Which of these separators will the Perceptron “prefer”?
Support Vector Machines (SVMs): “max. margin” linear classification
Hinge vs. Perceptron loss

Hinge loss upper bounds #mistakes; encourages "margin"

\[ \ell_H(w; x, y) = \max\{0, 1 - yw^Tx\} \]
SVM vs. Perceptron

- **Perceptron:**

\[ \hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \max\{0, -y_i w^T x_i\} \]

- **Support vector machine (SVM):**

\[ \hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \lambda \|w\|_2^2 \]
**Stochastic Gradient Descent**

- Start at an arbitrary \( w_0 \in \mathbb{R}^d \)
- For \( t=1,2,... \) do
  - Pick data point \( (x', y') \in D \) from training set uniformly at random (with replacement), and set
  \[
  w_{t+1} = w_t - \eta_t \nabla \ell(w_t; x', y')
  \]

- Hereby, \( \eta_t \) is called learning rate
- Guaranteed to converge under mild conditions, if
  \[
  \sum_t \eta_t = \infty \quad \text{and} \quad \sum_t \eta_t^2 < \infty
  \]
Support vector machines

- Widely used, very effective linear classifier
- Almost like Perceptron. Only differences:
  - Optimize slightly different, shifted loss (hinge loss)
  - Regularize weights (like ridge regression)
- Can optimize via stochastic gradient descent

- Safe choice for learning rate: \( \eta_t = \frac{1}{\lambda t} \)

- More details in Advanced Machine Learning lecture
SGD for SVM

\[ \hat{R}(w) = \sum_{i=1}^{n} \ell_H(w; x_i, y_i) + \frac{\lambda}{2} \|w\|_2^2 \]

\[ = \sum_{i=1}^{n} \left( \ell_H(w; x_i, y_i) + \frac{\lambda}{2} \|w\|_2^2 \right) f_i(w) \]

\[ \nabla_w \hat{R}(w) = \sum_{i=1}^{n} \nabla_w f_i(w) \]

\[ (\ast) = \nabla \ell_H(w) + \frac{\lambda}{2} \|w\|_2^2 \]

\[ \nabla \ell_H(w; x_i, y_i) = \begin{cases} \nabla \max(0, 1-y_i \langle w, x_i \rangle) & \text{if } y_i \langle w, x_i \rangle \geq 1 \\ -y_i x_i & \text{otherwise} \end{cases} \]

SGD: \[ w_{t+1} = w_t \left( 1 - \frac{\eta_t}{\lambda} \right) + \left[ y_i; \nabla w x_i < \right] y_i x_i \eta_t \]
Demo: Monitoring SGD
Choosing the regularization parameter

- Can pick regularization parameter via cross-validation just like in linear regression!
- Note that instead of using the hinge-loss for validation, would use the target performance metric (e.g., accuracy)
How can we find **nonlinear classification boundaries**?

Similar as in regression, can use **non-linear transformations** of the inputs as feature vectors.
Recall: linear regression for polynomials

- We can fit non-linear classifiers via linear methods, using nonlinear features of our data (basis functions)

\[ f(x) = \sum_{i=1}^{d} w_i \phi_i(x) \]

- For example: polynomials (in 1-D)

\[ f(x) = \sum_{i=0}^{m} w_i x^i \]

- Higher dimensions -> Monomials
Example

\[ x_1 = x_1, \quad x_2 = x^2 \]

\[ x_1 \rightarrow [x_1, x^2] \]
from sklearn.svm import LinearSVC

linearSVM = LinearSVC(C=1.0)
linearSVM.fit(X_train, y_train)
y_predict = linearSVM.predict(X_test)
Demo: Non-linear Classification with SVM
What you need to know

- The **Perceptron** is an algorithm for linear classification.
- It applies **Stochastic Gradient Descent (SGD)** on the Perceptron loss.
- **Mini-batches** exploit parallelism, reduce variance.
- The Perceptron loss is a **convex surrogate** function for the 0-1 (misclassification) loss.
- It is **guaranteed to produce a feasible solution** (a linear separator) if the data is separable.
- **SGD** is much more generally applicable.
- **Support Vector Machines (SVMs)** are closely related to Perceptron; use hinge loss and regularization.
Supervised learning big picture so far

- Least squares Regression
- Ridge Regression
- Perceptron
- Linear SVM

Loss function:
- Least squares Regression
- Ridge Regression
- Perceptron
- Linear SVM
## Supervised learning summary so far

<table>
<thead>
<tr>
<th>Representation/features</th>
<th>Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model/objective:</strong></td>
<td><strong>Loss-function + Regularization</strong></td>
</tr>
<tr>
<td></td>
<td>Squared loss, 0/1 loss, Perceptron loss, Hinge loss</td>
</tr>
<tr>
<td></td>
<td>$L^2$ norm</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>Exact solution, Gradient Descent, (mini-batch) SGD, Convex Programming, ...</td>
</tr>
<tr>
<td><strong>Evaluation metric:</strong></td>
<td>Mean squared error, Accuracy</td>
</tr>
<tr>
<td><strong>Model selection:</strong></td>
<td>K-fold Cross-Validation, Monte Carlo CV</td>
</tr>
</tbody>
</table>
Introduction to Machine Learning

Feature selection

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)
Feature selection

In many high-dimensional problems, we may prefer not to work with all potentially available features

Why?

- Interpretability (would like to “understand” the classifier, identify important variables/features)
- Generalization (simpler models may generalize better)
- Storage / computation / cost (don’t need to store / sum / acquire data for unused features...)

How?

- Naïve: try all subsets, and pick best (via crossvalidation)
Demo: Feature selection for regression
Greedy feature selection

- General purpose approach:
  - Greedily add (or remove) features to maximize
    - Cross-validated prediction accuracy
    - Mutual information or other notions of informativeness (not discussed)

- Can be used for *any* method
  (not only linear regression/classification)
Set of all features: \( V=\{1,\ldots,d\} \)

Define **cost function** for scoring subsets \( S \) of \( V \)

\( \hat{L}(S) \) is cross-validation error using features in \( S \) only

\[
X_i = [x_{i1}, \ldots, x_{id}] \quad \Rightarrow \quad X_{S,i} = [x_{i,j_1}, \ldots, x_{i,j_k}]
\]

Train model on

\[
\{(X_{S_{\bar{i}},1}, y_1), \ldots, (X_{S_{\bar{i}},n}, y_n)\}
\]

\( S = \{j_1, \ldots, j_k\}, j \neq i \)

\( \Rightarrow \hat{\omega}_S \)

\( \Rightarrow \hat{L}(S) \leftarrow \text{cross-validated performance of } \hat{\omega}_S \)
Greedy forward selection

- Start with $\mathcal{S} = \emptyset$ and $E_0 = \infty$
- For $i = 1:d$
  - find best element to add: $s_i = \arg \min_{j \in V \setminus \mathcal{S}} \hat{L}(\mathcal{S} \cup \{j\})$
  - compute error: $E_i = \hat{L}(\mathcal{S} \cup \{s_i\})$
  - If $E_i > E_{i-1}$ break, else set $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_i\}$
Problems with greedy forward selection

All classifiers using a single feature only have a 50% error rate! Can't tell informative from irrelevant features! [Diagram]
Demo: Forward Selection for Regression
Greedy backward selection

Start with $S = V$ and $E_{d+1} = \infty$

For $i = d:-1:1$

- find best element to remove: $s_i = \arg\min_{j \in S} \hat{L}(S \setminus \{j\})$

- compute error: $E_i = \hat{L}(S \setminus \{s_i\})$

- If $E_i > E_{i+1}$ break, else set $S \leftarrow S \setminus \{s_i\}$
Demo: Backward Selection for Regression
Comparison: FW vs. BW selection

<table>
<thead>
<tr>
<th>Method</th>
<th>Forward (FW)</th>
<th>Backward (BW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>Usually faster (if few relevant features)</td>
<td>Can handle „dependent“ features</td>
</tr>
</tbody>
</table>
Problems with greedy feature selection

- Computational cost (need to retrain models many times for different feature combinations)
- Can be suboptimal

Can we solve the learning & feature selection problem simultaneously via a single optimization?
Linear models: Feature selection = Sparsity

So far: explicitly select a subset of features

\[ x = [x_1, \ldots, x_d] \rightarrow x_S = [x_{i_1}, \ldots, x_{i_k}] \]

optimize over coefficients \( w_S = [w_{i_1}, \ldots, w_{i_k}] \)

\[ \hat{w}_S = \arg \min_{w_S} \sum_{i=1}^{n} (y_i - w_S^T x_{i,S})^2 \]

This is equivalent to constraining \( w \) to be sparse (i.e., contain at most \( k \) non-zero entries)
Joint feature selection and training

Would like to solve

$$\hat{w} = \text{arg min}_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \text{ s.t. } \|w\|_0 \leq k$$

where \( \|W\|_0 \) is the number of non-zeros in \( w \)

Alternatively, can penalize the number of nonzero entries:

$$\hat{w} = \text{arg min}_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_0$$
Making the optimization tractable

- Want to solve:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_0$$

- This is a difficult combinatorial optimization problem 😞
- Can view greedy algorithms before as heuristics for solving it

**Key idea:** Replace $\|\mathbf{W}\|_0$ by a more tractable term
L1 as surrogate for L0

$$\|\omega\|_1 = \sum_i |\omega_i|$$
The „sparsity trick“

\[ \|w\|_0 \quad \rightarrow \quad \|w\|_1 \]
Sparse regression: The Lasso

Before:

- Ridge regression

\[
\min_w \lambda \|w\|_2^2 + \sum_{i=1}^{n} (y_i - w^T x_i)^2
\]

- Uses \( \|w\|_2^2 \) to control the weights

Slight modification: replace \( \|w\|_2^2 \) by \( \|w\|_1 \)

- L1-regularized regression (the Lasso)

\[
\min_w \lambda \|w\|_1 + \sum_{i=1}^{n} (y_i - w^T x_i)^2
\]

This alternative penalty encourages coefficients to be exactly 0 ➞ automatic feature selection!
Lasso illustration

\[ \hat{w} = \text{argmin} \, \|w\|_1 + \lambda \sum (y_i - w^T x_i)^2 \]

\( \lambda \log, \; \text{assume} \; \|w\|_1 = 1 \)

\[ \exists w : \mathcal{F}(w) = \mathcal{F}(\hat{w}) \]
Lasso demo
Regularization paths

L2 (Ridge)

L1 (Lasso)
How to pick the regularization parameter?

Crossvalidation!
Another example: L1-SVM

- Before:
  - Support vector machine
    \[
    \min_{\mathbf{w}} \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)
    \]
  - Uses \(\|\mathbf{w}\|_2\) to control the weights
  - Apply sparsity trick: replace \(\|\mathbf{w}\|_2^2\) by \(\|\mathbf{w}\|_1\)
    - L1-SVM
      \[
      \min_{\mathbf{w}} \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)
      \]
  - This alternative penalty encourages coefficients to be exactly 0 \(\Rightarrow\) ignores those features!
Feature selection with L1-SVM

Experiment

Data:
- 38 train, 34 test data from a DNA microarray classification experiment (leukemia diagnosis)
- 7129 dimensions

<table>
<thead>
<tr>
<th>Method</th>
<th>CV Error</th>
<th>Test Error</th>
<th># of Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-norm SVM UR</td>
<td>2/38</td>
<td>3/34</td>
<td>22</td>
</tr>
<tr>
<td>2-norm SVM RFE</td>
<td>2/38</td>
<td>1/34</td>
<td>31</td>
</tr>
<tr>
<td>1-norm SVM</td>
<td>2/38</td>
<td>2/34</td>
<td>17</td>
</tr>
</tbody>
</table>

I1-SVM demo
Solving l1 regularized problems

- L1-norm is convex
- Combined with convex losses, obtain convex optimization problems (e.g., Lasso, l1-SVM, ...)
- Can in principle solve using (stochastic) gradient descent
- However, convergence usually slow, and will rarely get „exact 0“ entries
- Much recent work in convex optimization deals with solving such problems very efficiently
  - Proximal methods (not discussed in this class)
## Comparison: Greedy selection vs. L1-Regularization

<table>
<thead>
<tr>
<th>Method</th>
<th>Greedy (FW/BW)</th>
<th>L1-Regularization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>Applies to any prediction method</td>
<td>Faster (training and feature selection happen jointly)</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>Slower (need to train many models)</td>
<td>Only works for linear models</td>
</tr>
</tbody>
</table>
What do you need to know

- What is feature selection
- Greedy algorithm (forward and backward)
- $l_1$-regularization to encourage sparsity
  - Example: The Lasso ($l_1$-regression)
  - Example: $l_1$-SVM
- Advantages and disadvantages of the respective methods
## Supervised learning summary so far

<table>
<thead>
<tr>
<th>Representation/features</th>
<th>Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model/objective:</strong></td>
<td><strong>Loss-function</strong> + <strong>Regularization</strong></td>
</tr>
<tr>
<td></td>
<td>Squared loss, 0/1 loss, Perceptron loss, Hinge loss</td>
</tr>
<tr>
<td></td>
<td><em>L</em>₂ norm, <em>L</em>₁ norm, <em>L</em>₀ penalty</td>
</tr>
<tr>
<td><strong>Method:</strong></td>
<td>Exact solution, Gradient Descent, (mini-batch) SGD, <strong>Greedy selection</strong>, Convex Programming, ...</td>
</tr>
<tr>
<td><strong>Evaluation metric:</strong></td>
<td>Mean squared error, Accuracy</td>
</tr>
<tr>
<td><strong>Model selection:</strong></td>
<td>K-fold Cross-Validation, Monte Carlo CV</td>
</tr>
</tbody>
</table>
Supervised learning big picture so far

- Least squares Regression
- Ridge Regression
- Perceptron
- L1-Regression (Lasso)
- Linear SVM
- L1-SVM
- L1-regularizer
- L2-regularizer

Loss function.