Course organization

- **Retrieval**
  - Given a query, find “most similar” item in a large data set
  - *Applications*: GoogleGoggles, Shazam, ...

- **Supervised learning** (Classification, Regression)
  - Learn a concept (function mapping queries to labels)
  - *Applications*: Spam filtering, predicting price changes, ...

- **Unsupervised learning** (Clustering, dimension reduction)
  - Identify clusters, “common patterns”; anomaly detection
  - *Applications*: Recommender systems, fraud detection, ...

- **Learning with limited feedback**
  - Learn to optimize a function that’s expensive to evaluate
  - *Applications*: Online advertising, opt. UI, learning rankings, ...
How can we solve this optimization?

What about local minima?

This is a convex (quadratic) program
Generally: Online convex programming

- **Input:**
  - Feasible set \( S \subseteq \mathbb{R}^d \)
  - Starting point \( w_0 \in S \)

- **Each round** \( t \) **do**
  - Receive convex function \( f_t : S \rightarrow \mathbb{R} \)
  - Incur loss \( \ell_t = f_t(w_t) \)
  - Update:
    \[
    w_{t+1} = \text{Proj}_S(w_t - \eta_t \nabla f_t(w_t))
    \]

- **Regret:**
  \[
  R_T = \left( \sum_{t=1}^{T} \ell_t \right) - \min_{w \in S} \sum_{t=1}^{T} f_t(w)
  \]
Theorem [Zinkevich ‘03]

Let $f_1, \ldots, f_T$ be an arbitrary sequence of convex functions with feasible set $S$

Set $\eta_t = 1/\sqrt{t}$

Then, the regret of online convex programming is bounded by

$$ R_T \leq \frac{||S||^2 \sqrt{T}}{2} + \left( \sqrt{T} - \frac{1}{2} \right) ||\nabla f||^2 $$

additional loss in accuracy due to online setting

$$ \frac{R_T}{T} = O\left(\frac{1}{\sqrt{T}}\right) = O\left(\frac{1}{\sqrt{T}}\right) \rightarrow 0 $$
More results on supervised learning

- Feature selection
- Dealing with multiple classes
- **Regression**
- Nonlinear methods
Regression

- So far, our goal was to predict a discrete label.
- In many problems, we need to predict a real-valued output.

\[ y = f(x; w) + noise \]

- E.g.:
  - Predict grade based on #homeworks solved.
  - Predict flight delay at one airport given delays at other airports.
  - ...
Given \((x_1, y_1), \ldots, (x_n, y_n)\)

Assume: \(y_i = w^T x_i + \text{noise}\)

To optimize \(w\) need to quantify goodness of fit
Square loss

- Want to solve

\[ \begin{align*}
  x_i & \in \mathbb{R}^d, \\
  X & = \begin{pmatrix}
    x_1^T \\
    x_2^T \\
    \vdots \\
    x_n^T
  \end{pmatrix} \in \mathbb{R}^{N \times d}
\end{align*} \]

\[ w^* = \arg \min_w \sum_{i=1}^n (y_i - w^T x_i)^2 \]

- Closed form solution:

\[ w^* = (X^T X)^{-1} X^T y \]

- Complexity?

\[ \mathcal{O}(d^3) \]

Also, maybe no unique solution?

- Intractable for large # of dimensions!

- Will see how we can efficiently compute with OCP!
Learning non-linear functions

Key insight: Can learn nonlinear functions using linear methods! Works for classification too!
Solving nonlinear problems

\[ \text{s.t.} \quad \sum_{i=1}^{D} \phi_i(x) w_i + b = 0 \]

where \( \phi(x) \in \mathbb{R}^D \) is a nonlinear transformation.

### Example

\[ \phi(x) = (x_1, x_1 x_2, x_2^2) \in \mathbb{R}^3 \]

The problem becomes linearly separable in the higher-dimensional feature space.
Suppose we consider polynomials. Which degree should we choose?
When learning complex / high dimensional functions, need to control the complexity of the model

- In practice, this means ensuring that weights $w$ are small

- This process is called *regularization*
Regularized regression

• Ridge regression:

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

• Closed form solution: ensures a unique solution

\[
w^* = (X^TX + \lambda I)^{-1}X^Ty
\]

• Shrinks weights of ‘unimportant’ variables.
Regularized regression

- L1-regularized regression – “Lasso”:

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

- In general, no closed form solution.

  \[ \text{If } X^T X = I \]
  \[ w^* = \text{sign}(w)(\|w\|_1 - \lambda)^+ \]
More general loss functions

A large fraction of methods in supervised learning can be reduced to optimization problems of the form

\[ w^* = \arg \min_w \lambda ||w|| + \sum_{i=1}^{n} \ell(y_i; x_i, w_i) \]

Example loss functions

- Hinge loss (SVM)
- Multi-class hinge loss
- Log loss (next homework!)
- Square loss
- \( \varepsilon \)-sensitive loss
- ...

15
Solving regularized learning problems

- Reduce to online convex programming:

\[ w_{t+1} = \text{Proj}_S(x_t - \eta_t \nabla \ell(y_t; x_t, w_t)) \]

- Gradient computation specific to loss function

- Reprojection: Need to solve

\[ \arg \min_{w' \in S} \| w' - w_t \|_2 \]
Choosing the right regularizer

- How should we choose the regularization parameter?

\[ \text{Strength of regularizer } \lambda \]

- Error vs. Strength of regularizer \( \lambda \)
Cross-validation

- May overfit if we optimize for fixed training set!
- Remedy: Cross-validation

\[
\begin{array}{c|c|c|c}
D_1 & D_2 & \ldots & D_k \\
\end{array}
\]

- Split data set into k “folds”
- For each possible regularization parameter setting \( \lambda \):
  - For \( i = 1:k \)
    - Train on all but \( i \)-th fold; calculate error \( E_i \)
    - Estimate generalization error for param. \( \lambda \) as

\[
\frac{1}{k} \sum_i E_i
\]
Aside: Cross-validation

- How to choose k? 5, 10...

\[ K = n \left(1 - \frac{1}{\log n - 1}\right) \]

- Then cross-validation is equivalent to the Bayesian Information Criterion

\[ BIC = -2 \log \ell + m \log n \]

- CV penalises the *degrees of freedom*.
- These results only apply for *linear models* with *squared error loss*. 
Aside: Dual formulation of SVM

- **Primal form:**
  \[
  \min_{w,b,\xi \geq 0} \ w^T w + C \sum \xi_i \\
  \text{s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i
  \]

  Using Lagrange multipliers:

  \[
  \min_{w,b,\xi \geq 0} \ \max_{\alpha \geq 0} \ w^T w + C \sum \xi_i - \sum \alpha_i[y_i(w^T x_i + b) - 1 + \xi_i] - \sum \lambda_i \xi_i
  \]

- **Dual form:**

  \[
  \begin{align*}
  \frac{\partial L}{\partial b} &= 0 \rightarrow \sum \alpha_i y_i = 0 \\
  \frac{\partial L}{\partial w} &= 0 \rightarrow w = \sum \alpha_i y_i x_i \\
  \frac{\partial L}{\partial \xi_i} &= 0 \rightarrow C - \alpha_i - \lambda_i = 0 \\
  \frac{\partial L}{\partial \alpha_i} &= 0 \rightarrow 0 \leq \alpha_i \leq C
  \end{align*}
  \]

  \[
  \max \sum \alpha_i - \frac{1}{2} \sum \alpha_i \alpha_j y_i y_j x_i^T x_j
  \]

  \[
  0 \leq \alpha_i \leq C
  \]
Aside: The „Kernel Trick“

- Standard lesson in Machine Learning:
  - Can solve linear problem in feature space implicitly using inner products only
    \[ \phi(x) = (x_1^2, x_1 x_2, x_2^2) \]

- Example: Dual formulation of SVM
  \[
  \max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j)
  \]
  s.t. \(0 \leq \alpha_i \leq C\) and \(\sum_{i=1}^{N} \alpha_i y_i = 0\)

  \[
  w = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)
  \]

  Can we use this for large data??

\[
\beta x + \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)
\]
Idea: **Explicitly** generate low-dim (nonlinear!) features

Construct \( z(x) \in \mathbb{R}^{d'} \)

\[ z(x)^T z(x') \approx k(x, x') \]

do linear classification / regression explicitly with \( z(x) \)

\[ \phi(x) \in \mathbb{R}^D \]
Algorithm 1 Random Fourier Features.

Require: A positive definite shift-invariant kernel \( k(x, y) = k(x - y) \).
Ensure: A randomized feature map \( z(x) : \mathcal{R}^d \rightarrow \mathcal{R}^{2D} \) so that \( z(x)'z(y) \approx k(x - y) \).

Compute the Fourier transform \( p \) of the kernel \( k \):
\[
p(\omega) = \frac{1}{2\pi} \int e^{-j\omega'\Delta} k(\Delta) \, d\Delta.
\]

Draw \( D \) iid samples \( \omega_1, \cdots, \omega_D \in \mathcal{R}^d \) from \( p \).

Let \( z(x) \equiv \sqrt{\frac{1}{D}} \left[ \cos(\omega_1'x) \cdots \cos(\omega_D'x) \sin(\omega_1'x) \cdots \sin(\omega_D'x) \right]' \).

<table>
<thead>
<tr>
<th>Kernel Name</th>
<th>( k(\Delta) )</th>
<th>( p(\omega) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian</td>
<td>( e^{-\frac{|\Delta|^2}{2}} )</td>
<td>( (2\pi)^{-\frac{D}{2}} e^{-\frac{|\omega|^2}{2}} )</td>
</tr>
<tr>
<td>Laplacian</td>
<td>( e^{-|\Delta|_1} )</td>
<td>( \prod_d \frac{1}{\pi(1+\omega_d^2)} )</td>
</tr>
<tr>
<td>Cauchy</td>
<td>( \prod_d \frac{2}{1+\Delta_d^2} )</td>
<td>( e^{-|\Delta|_1} )</td>
</tr>
</tbody>
</table>
Performance of random features

**Claim 1** (Uniform convergence of Fourier features). *Let $\mathcal{M}$ be a compact subset of $\mathbb{R}^d$ with diameter $\text{diam}(\mathcal{M})$. Then, for the mapping $z$ defined in Algorithm 1, we have*

$$\Pr \left[ \sup_{x, y \in \mathcal{M}} |z(x)'z(y) - k(x, y)| \geq \epsilon \right] \leq 2^8 \left( \frac{\sigma_p \text{diam}(\mathcal{M})}{\epsilon} \right)^2 \exp \left( -\frac{D\epsilon^2}{4(d + 2)} \right),$$

*where $\sigma_p^2 \equiv E_p[\omega'\omega]$ is the second moment of the Fourier transform of $k$. Further, $\sup_{x, y \in \mathcal{M}} |z(x)'z(y) - k(y, x)| \leq \epsilon$ with any constant probability when $D = \Omega \left( \frac{d}{\epsilon^2} \log \frac{\sigma_p \text{diam}(\mathcal{M})}{\epsilon} \right)$.*

- Solving linear SVM on explicit (random) features provably „almost the same“ as solving non-linear SVM
Performance of random features [RR ‘07]

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fourier+LS</th>
<th>Binning+LS</th>
<th>CVM</th>
<th>Exact SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU regression</td>
<td>3.6%</td>
<td>5.3%</td>
<td>5.5%</td>
<td>11%</td>
</tr>
<tr>
<td>6500 instances 21 dims</td>
<td>20 secs</td>
<td>3 mins</td>
<td>51 secs</td>
<td>31 secs</td>
</tr>
<tr>
<td></td>
<td>$D = 300$</td>
<td>$P = 350$</td>
<td></td>
<td>ASVM</td>
</tr>
<tr>
<td>Census regression</td>
<td>5%</td>
<td>7.5%</td>
<td>8.8%</td>
<td>9%</td>
</tr>
<tr>
<td>18,000 instances 119 dims</td>
<td>36 secs</td>
<td>19 mins</td>
<td>7.5 mins</td>
<td>13 mins</td>
</tr>
<tr>
<td></td>
<td>$D = 500$</td>
<td>$P = 30$</td>
<td></td>
<td>SVM Torch</td>
</tr>
<tr>
<td>Adult classification</td>
<td>14.9%</td>
<td>15.3%</td>
<td>14.8%</td>
<td>15.1%</td>
</tr>
<tr>
<td>32,000 instances 123 dims</td>
<td>9 secs</td>
<td>1.5 mins</td>
<td>73 mins</td>
<td>7 mins</td>
</tr>
<tr>
<td></td>
<td>$D = 500$</td>
<td>$P = 30$</td>
<td></td>
<td>SVM light</td>
</tr>
<tr>
<td>Forest Cover classification</td>
<td>11.6%</td>
<td>2.2%</td>
<td>2.3%</td>
<td>2.2%</td>
</tr>
<tr>
<td>522,000 instances 54 dims</td>
<td>71 mins</td>
<td>25 mins</td>
<td>7.5 hrs</td>
<td>44 hrs</td>
</tr>
<tr>
<td></td>
<td>$D = 5000$</td>
<td>$P = 50$</td>
<td></td>
<td>libSVM</td>
</tr>
<tr>
<td>KDDCUP99 (see footnote)</td>
<td>7.3%</td>
<td>7.3%</td>
<td>6.2% (18%)</td>
<td>8.3%</td>
</tr>
<tr>
<td>classification</td>
<td>1.5 min</td>
<td>35 mins</td>
<td>1.4 secs (20 secs)</td>
<td>&lt; 1 s</td>
</tr>
<tr>
<td>4,900,000 instances 127 dims</td>
<td>1.5 min</td>
<td>$P = 10$</td>
<td></td>
<td>SVM+sampling</td>
</tr>
</tbody>
</table>

- Linear SVM/Regression on random features outperforms nonlinear methods
Online convex programming is a natural approach to solve regularized learning problems.

Can be parallelized (to some extent).

Flexible choice of loss function and regularizer gives rise to many useful methods:
- SVM
- L1-SVM
- Ridge regression
- L1-regularized regression
- Logistic regression (homework)
- ...

Can even learn nonlinear functions!