Regrade requests submitted directly in Gradescope, do not email instructors.

For each block compute the memory required in terms of $n$, $p$, $d$.

If $d \ll p \ll n$, what is the most memory efficient program (blue, green, red)?

If you have unlimited memory, what do you think is the fastest program?

```python
# generate some nonsense data for an example
X = np.random.randn(n,d)
y = np.random.randn(n)

# generate the random features
G = np.random.randn(p, d)*np.sqrt(.1)
b = np.random.randn(p)*2*np.pi

H = np.dot(X, G.T) + b.T
HTH = np.dot(H.T, H)
HTy = np.dot(H.T, y)

# construct HTH
HTH = np.zeros((p,p))
HTy = np.zeros(p)
for i in range(n):
    hi = np.dot(X[i,:], G.T)+b
    HTH += np.outer(hi, hi)
    HTy += y[i]*hi
if i % 1000==0: print(i)

w = np.linalg.solve(HTH + lam*np.eye(p), HTy)
```

1 float in NumPy = 8 bytes
$10^6 \approx 2^{20}$ bytes = 1 MB
$10^9 \approx 2^{30}$ bytes = 1 GB
Gradient Descent

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 18, 2016
Machine Learning Problems

- Have a bunch of iid data of the form:

\[
\{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}
\]

- Learning a model’s parameters:

Each \(\ell_i(w)\) is convex.

\[
\sum_{i=1}^{n} \ell_i(w)
\]
Machine Learning Problems

- Have a bunch of iid data of the form:
  \[ \{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:
  Each \( \ell_i(w) \) is convex.

\[ \sum_{i=1}^{n} \ell_i(w) \]

\( g \) is a subgradient at \( x \) if
\[ f(y) \geq f(x) + g^T(y - x) \]

\( f \) convex:
\[ f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \quad \forall x, y, \lambda \in [0, 1] \]
\[ f(y) \geq f(x) + \nabla f(x)^T(y - x) \quad \forall x, y \]
Machine Learning Problems

- Have a bunch of iid data of the form:
  \[ \{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:
  Each \( \ell_i(w) \) is convex.

  \[ \sum_{i=1}^{n} \ell_i(w) \]

  Logistic Loss: \( \ell_i(w) = \log(1 + \exp(-y_i x_i^T w)) \)

  Squared error Loss: \( \ell_i(w) = (y_i - x_i^T w)^2 \)
Least squares

- Have a bunch of iid data of the form:

\[ \{ (x_i, y_i) \}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:

  Each \( \ell_i(w) \) is convex. 

  Squared error Loss: \( \ell_i(w) = (y_i - x_i^T w)^2 \)

  How does software solve:

  \[ \frac{1}{2} ||Xw - y||_2^2 \]
Least squares

- Have a bunch of iid data of the form:
  \[
  \{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}
  \]

- Learning a model’s parameters:

  Each \( \ell_i(w) \) is convex.

  Squared error Loss: \( \ell_i(w) = (y_i - x_i^T w)^2 \)

How does software solve:

\[
\frac{1}{2} \|Xw - y\|_2^2
\]

...its complicated:

Do you need high precision?
Is \( X \) column/row sparse?
Is \( \hat{w}_{LS} \) sparse?
Is \( X^T X \) “well-conditioned”?
Can \( X^T X \) fit in cache/memory?

(LAPACK, BLAS, MKL...)
Taylor Series Approximation

- Taylor series in one dimension:
  \[ f(x + \delta) = f(x) + f'(x)\delta + \frac{1}{2} f''(x)\delta^2 + \ldots \]

- Gradient descent:
Taylor Series Approximation

- Taylor series in $d$ dimensions:

$$f(x + v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v + \ldots$$

- Gradient descent:
Gradient Descent

\[ f(w) = \frac{1}{2} \|Xw - y\|_2^2 \]

\[ w_{t+1} = w_t - \eta \nabla f(w_t) \]

\[ \nabla f(w) = \]
Gradient Descent

\[ f(w) = \frac{1}{2} ||Xw - y||^2 \]

\[ w_{t+1} = w_t - \eta \nabla f(w_t) \]

\[ \nabla f(w) = X^T(Xw - y) \]

\[ w_* = \text{arg min}_w f(w) \implies \nabla f(w_*) = 0 \]

\[ w_{t+1} - w_* = w_t - w_* - \eta \nabla f(w_t) \]

\[ = w_t - w_* - \eta(\nabla f(w_t) - \nabla f(w_*)) \]

\[ = w_t - w_* - \eta X^T X (w_t - w_*) \]

\[ = (I - \eta X^T X)(w_t - w_*) \]

\[ = (I - \eta X^T X)^{t+1}(w_0 - w_*) \]
Gradient Descent

\[ f(w) = \frac{1}{2} \| Xw - y \|_2^2 \]

\[ w_{t+1} = w_t - \eta \nabla f(w_t) \]

\[ (w_{t+1} - w_*) = (I - \eta X^T X)(w_t - w_*) \]

\[ = (I - \eta X^T X)^{t+1}(w_0 - w_*) \]

Example:

\[
\begin{bmatrix}
10^{-3} & 0 \\
0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
10^{-3} \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

\[ w_0 = \]

\[ w_* = \]
Gradient Descent

\[ f(w) = \frac{1}{2} \|Xw - y\|_2^2 \]

\[ w_{t+1} = w_t - \eta \nabla f(w_t) \]

\[ (w_{t+1} - w_*) = (I - \eta X^T X)(w_t - w_*) \]

\[ = (I - \eta X^T X)^{t+1}(w_0 - w_*) \]

Example:

\[ X = \begin{bmatrix} 10^{-3} & 0 \\ 0 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 10^{-3} \\ 1 \end{bmatrix} \quad w_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad w_* = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]

\[ X^T X = \begin{bmatrix} 10^{-6} & 0 \\ 0 & 1 \end{bmatrix} \]

Pick \( \eta \) such that

\[ \max\{|1 - \eta 10^{-6}|, |1 - \eta|\} < 1 \]

\[ |w_{t+1,1} - w_{*,1}| = |1 - \eta 10^{-6}|^{t+1} \quad |w_{0,1} - w_{*,1}| = |1 - \eta 10^{-6}|^{t+1} \]

\[ |w_{t+1,2} - w_{*,2}| = |1 - \eta|^{t+1} \quad |w_{0,2} - w_{*,2}| = |1 - \eta|^{t+1} \]
Taylor Series Approximation

- Taylor series in one dimension:

\[ f(x + \delta) = f(x) + f'(x)\delta + \frac{1}{2} f''(x)\delta^2 + \ldots \]

- Newton’s method:
Taylor Series Approximation

- Taylor series in $d$ dimensions:

\[ f(x + v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v + \ldots \]

- Newton’s method:
Newton’s Method \[ f(w) = \frac{1}{2} ||Xw - y||^2_2 \]

\[ \nabla f(w) = \]

\[ \nabla^2 f(w) = \]

\( v_t \) is solution to: \[ \nabla^2 f(w_t) v_t = -\nabla f(w_t) \]

\[ w_{t+1} = w_t + \eta v_t \]
Newton’s Method  \[ f(w) = \frac{1}{2} \|Xw - y\|^2 \]

\[ \nabla f(w) = X^T (Xw - y) \]
\[ \nabla^2 f(w) = X^T X \]

\( v_t \) is solution to: \[ \nabla^2 f(w_t)v_t = -\nabla f(w_t) \]

\[ w_{t+1} = w_t + \eta v_t \]

For quadratics, Newton’s method can converge in one step! (No surprise, why?)

\[ w_1 = w_0 - \eta (X^T X)^{-1} X^T (Xw_0 - y) \]
\[ = (1 - \eta)w_0 + \eta (X^T X)^{-1} X^T y \]
\[ = (1 - \eta)w_0 + \eta w_* \]

In general, for \( w_t \) “close enough” to \( w_* \) one should use \( \eta = 1 \)
In general for Newton’s method to achieve $f(w_t) - f(w_*) \leq \epsilon$:

So why are ML problems overwhelmingly solved by gradient methods?

Hint: $v_t$ is solution to: $\nabla^2 f(w_t) v_t = -\nabla f(w_t)$
General Convex case \( f(w_t) - f(w_*) \leq \epsilon \)

Newton’s method:
\[ t \approx \log(\log(1/\epsilon)) \]

Gradient descent:
- f is smooth and strongly convex: \( aI \leq \nabla^2 f(w) \leq bI \)
- f is smooth: \( \nabla^2 f(w) \leq bI \)
- f is potentially non-differentiable: \( \|\nabla f(w)\|_2 \leq c \)

Other: BFGS, Heavy-ball, BCD, SVRG, ADAM, Adagrad,…

Nocedal + Wright, Bubeck

Clean converge nice proofs: Bubeck
Revisiting… Logistic Regression

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 18, 2016
Loss function: Conditional Likelihood

- Have a bunch of iid data of the form: \( \{(x_i, y_i)\}_{i=1}^n \) \( x_i \in \mathbb{R}^d, \ y_i \in \{-1, 1\} \)

\[
\hat{w}_{MLE} = \arg \max_w \prod_{i=1}^n P(y_i | x_i, w) \quad P(Y = y|x, w) = \frac{1}{1 + \exp(-y w^T x)}
\]

\[
f(w) = \arg \min_w \sum_{i=1}^n \log(1 + \exp(-y_i x_i^T w))
\]

\[
\nabla f(w) =
\]
Stochastic Gradient Descent

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 18, 2016
Stochastic Gradient Descent

- Have a bunch of iid data of the form:
  \[
  \{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}
  \]

- Learning a model’s parameters:
  Each \( \ell_i(w) \) is convex.
  \[
  \frac{1}{n} \sum_{i=1}^n \ell_i(w)
  \]
Stochastic Gradient Descent

- Have a bunch of iid data of the form:
  \[ \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^d, \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:
  Each \( \ell_i(w) \) is convex.

Gradient Descent:
\[
w_{t+1} = w_t - \eta \nabla_w \left( \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \right) \bigg|_{w=w_t}
\]
Stochastic Gradient Descent

- Have a bunch of iid data of the form:

  \[ \{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:

  Each \( \ell_i(w) \) is convex.

**Gradient Descent:**

\[
w_{t+1} = w_t - \eta \nabla_w \left( \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \right) \bigg|_{w=w_t}
\]

**Stochastic Gradient Descent:**

\[
w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \bigg|_{w=w_t}
\]

\[I_t \text{ drawn uniform at random from } \{1, \ldots, n\}\]

\[\mathbb{E}[\nabla \ell_{I_t}(w)] = \]
Stochastic Gradient Descent

Theorem

Let

\[ w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \bigg|_{w=w_t} \]

so that

\[ \mathbb{E} [\nabla \ell_{I_t}(w)] = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(w) =: \nabla \ell(w) \]

If

\[ \|w_1 - w_0\|_2^2 \leq R \]

and

\[ \sup_w \max_i \|\nabla \ell_i(w)\|_2 \leq G \]

then

\[ \mathbb{E} [\ell(\bar{w}) - \ell(w_\ast)] \leq \frac{R}{2T\eta} + \frac{\eta G}{2} \leq \sqrt{\frac{RG}{GT}} \]

\[ \eta = \sqrt{\frac{R}{GT}} \]

\[ \bar{w} = \frac{1}{T} \sum_{t=1}^{T} w_t \]

(In practice use last iterate)
Stochastic Gradient Descent

Proof

$$\mathbb{E}[||w_{t+1} - w_*||^2_2] = \mathbb{E}[||w_t - \eta \nabla \ell_{I_t}(w_t) - w_*||^2_2]$$
Stochastic Gradient Descent

Proof

\[
\mathbb{E}[||w_{t+1} - w_*||^2_2] = \mathbb{E}[||w_{t} - \eta \nabla \ell_{I_t}(w_t) - w_*||^2_2]
\]
\[
= \mathbb{E}[||w_t - w_*||^2_2] - 2\eta \mathbb{E}[(\nabla \ell_{I_t}(w_t))^T(w_t - w_*)] + \eta^2 \mathbb{E}[||\nabla \ell_{I_t}(w_t)||^2_2]
\]
\[
\leq \mathbb{E}[||w_t - w_*||^2_2] - 2\eta \mathbb{E}[\ell(w_t) - \ell(w_*)] + \eta^2 G
\]

\[
\mathbb{E}[\nabla \ell_{I_t}(w_t)^T(w_t - w_*)] = \mathbb{E}[\mathbb{E}[\nabla \ell_{I_t}(w_t)^T(w_t - w_*)|I_1, w_1, \ldots, I_{t-1}, w_{t-1}]]
\]
\[
= \mathbb{E}[\nabla \ell(w_t)^T(w_t - w_*)]
\]
\[
\geq \mathbb{E}[\ell(w_t) - \ell(w_*)]
\]

\[
\sum_{t=1}^{T} \mathbb{E}[\ell(w_t) - \ell(w_*)] \leq \frac{1}{2\eta} \left( \mathbb{E}[||w_1 - w_*||^2_2] - \mathbb{E}[||w_{T+1} - w_*||^2_2] + T\eta^2 G \right)
\]
\[
\leq \frac{R}{2\eta} + \frac{T\eta G}{2}
\]
Stochastic Gradient Descent

Proof

Jensen’s inequality:
For any random $Z \in \mathbb{R}^d$ and convex function $\phi : \mathbb{R}^d \to \mathbb{R}$, $\phi(\mathbb{E}[Z]) \leq \mathbb{E}[\phi(Z)]$

$$\mathbb{E}[\ell(\bar{w}) - \ell(w_*)] \leq \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\ell(w_t) - \ell(w_*)]$$

$$\bar{w} = \frac{1}{T} \sum_{t=1}^{T} w_t$$
Proof

**Jensen’s inequality:**
For any random \( Z \in \mathbb{R}^d \) and convex function \( \phi : \mathbb{R}^d \to \mathbb{R} \), \( \phi(\mathbb{E}[Z]) \leq \mathbb{E}[\phi(Z)] \)

\[
\mathbb{E}[\ell(\bar{w}) - \ell(w^*)] \leq \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\ell(w_t) - \ell(w^*)] \\
\bar{w} = \frac{1}{T} \sum_{t=1}^{T} w_t
\]

\[
\mathbb{E}[\ell(\bar{w}) - \ell(w^*)] \leq \frac{R}{2T\eta} + \frac{\eta G}{2} \leq \sqrt{\frac{RG}{2T}} \\
\eta = \sqrt{\frac{R}{GT}}
\]
Learning Problems as Expectations

- Minimizing loss in training data:
  - Given dataset:
    - Sampled iid from some distribution \( p(x) \) on features:
  - Loss function, e.g., hinge loss, logistic loss,…
  - We often minimize loss in training data:

\[
\ell_D(w) = \frac{1}{N} \sum_{j=1}^{N} \ell(w, x_j)
\]

- However, we should really minimize expected loss on all data:

\[
\ell(w) = E_x [\ell(w, x)] = \int p(x) \ell(w, x) dx
\]

- So, we are approximating the integral by the average on the training data
Gradient descent in Terms of Expectations

- “True” objective function:
\[ \ell(w) = E_x [\ell(w, x)] = \int p(x) \ell(w, x) dx \]

- Taking the gradient:

- “True” gradient descent rule:

- How do we estimate expected gradient?
SGD: Stochastic Gradient Descent

- "True" gradient: \( \nabla \ell(w) = E_x [\nabla \ell(w, x)] \)

- Sample based approximation:

- What if we estimate gradient with just one sample???
  - Unbiased estimate of gradient
  - Very noisy!
  - **Also** called stochastic gradient descent
    - Among many other names
  - **VERY** useful in practice!!!
Click prediction for ads is a streaming data task:

- User enters query, and ad must be selected
  - Observe $x_i$, and must predict $y_i$

- User either clicks or doesn’t click on ad
  - Label $y_i$ is revealed afterwards
    - Google gets a reward if user clicks on ad

- Update model for next time
Online classification

New point arrives at time $k$
The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction:

- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe features:
    - Make prediction:
    - Observe true class:
      - Update model:
        - If prediction is not equal to truth
The Perceptron Algorithm

Classification setting: $y$ in $\{-1,+1\}$

Linear model
- Prediction: $\text{sign}(w^T x_i + b)$

Training:
- Initialize weight vector: $w_0 = 0, b_0 = 0$
- At each time step:
  - Observe features: $x_k$
  - Make prediction: $\text{sign}(x_k^T w_k + b_k)$
  - Observe true class: $y_k$
- Update model:
  - If prediction is not equal to truth

$$
\begin{bmatrix}
w_{k+1} \\
b_{k+1}
\end{bmatrix} = 
\begin{bmatrix}
w_k \\
b_k
\end{bmatrix} + y_k 
\begin{bmatrix}
x_k \\
1
\end{bmatrix}
$$
"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

*The New York Times, 1958*
Linear Separability

- Perceptron guaranteed to converge if
  - Data linearly separable:
Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples:
  - Each feature vector has bounded norm:
  - If dataset is linearly separable:

- Then the number of mistakes made by the online perceptron on any such sequence is bounded by
Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it’s done for ever!
    - Even if you see infinite data
Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it’s done for ever!
    - Even if you see infinite data

- Perceptron is useless in practice!
  - Real world not linearly separable
  - If data not separable, cycles forever and hard to detect
  - Even if separable may not give good generalization accuracy (small margin)
What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood

- When we discussed the Perceptron:
  - Started from description of an algorithm

- What is the Perceptron optimizing????
Support Vector Machines

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 18, 2018
Linear classifiers – Which line is better?
Pick the one with the largest margin!

\[ x^Tw + b = 0 \]

margin \(2\gamma\)
Pick the one with the largest margin!

Distance from $x_0$ to hyperplane defined by $x^T w + b = 0$?
Pick the one with the largest margin!

Distance from $x_0$ to hyperplane defined by $x^T w + b = 0$?

If $\tilde{x}_0$ is the projection of $x_0$ onto the hyperplane then

$$||x_0 - \tilde{x}_0||_2 = |(x_0^T - \tilde{x}_0)^T \frac{w}{||w||_2}|$$

$$= \frac{1}{||w||_2} |x_0^T w - \tilde{x}_0^T w|$$

$$= \frac{1}{||w||_2} |x_0^T w + b|$$
Pick the one with the largest margin!

Distance of $x_0$ from hyperplane $x^T w + b$:
$$\frac{1}{||w||_2} (x_0^T w + b)$$

Optimal Hyperplane

$$\max_{w,b} \gamma$$
subject to
$$\frac{1}{||w||_2} y_i(x_i^T w + b) \geq \gamma \quad \forall i$$
Pick the one with the largest margin!

Distance of $x_0$ from hyperplane $x^T w + b$:
$$\frac{1}{\|w\|_2}(x_0^T w + b)$$

Optimal Hyperplane
$$\max_{w,b} \gamma$$
subject to
$$\frac{1}{\|w\|_2} y_i(x_i^T w + b) \geq \gamma \quad \forall i$$

Optimal Hyperplane (reparameterized)
$$\min_{w,b} \|w\|_2^2$$
subject to
$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$
Pick the one with the largest margin!

- Solve efficiently by many methods, e.g.,
  - quadratic programming (QP)
    - Well-studied solution algorithms
  - Stochastic gradient descent
  - Coordinate descent (in the dual)

\[
\begin{align*}
\min_{w,b} & \quad \|w\|_2^2 \\
\text{subject to} & \quad y_i(x_i^T w + b) \geq 1 \quad \forall i
\end{align*}
\]
What if the data is still not linearly separable?

If data is linearly separable

\[
\min_{w,b} \frac{1}{2} \|w\|_2^2 \quad \text{subject to} \quad y_i (x_i^T w + b) \geq 1 \quad \forall i
\]
What if the data is still not linearly separable?

- If data is linearly separable
  \[
  \min_{w,b} \|w\|_2^2 \\
  y_i(x_i^T w + b) \geq 1 \quad \forall i
  \]

- If data is not linearly separable, some points don’t satisfy margin constraint:
  \[
  \min_{w,b} \|w\|_2^2 \\
  y_i(x_i^T w + b) \geq 1 - \xi_i \quad \forall i \\
  \xi_i \geq 0, \sum_{j=1}^{n} \xi_j \leq \nu
  \]
What if the data is still not linearly separable?

- If data is linearly separable
  \[
  \min_{w,b} \frac{1}{||w||_2^2} \quad \text{s.t.} \quad y_i(x_i^T w + b) \geq 1 \quad \forall i
  \]

- If data is not linearly separable, some points don’t satisfy margin constraint:
  \[
  \min_{w,b} \frac{1}{||w||_2^2} \quad \text{s.t.} \quad y_i(x_i^T w + b) \geq 1 - \xi_i \quad \forall i
  \]
  \[
  \xi_i \geq 0, \sum_{j=1}^{n} \xi_j \leq \nu
  \]

- What are “support vectors?”
SVM as penalization method

- Original quadratic program with linear constraints:

\[
\begin{align*}
\min_{w,b} & \quad \|w\|^2_2 \\
y_i(x_i^Tw + b) & \geq 1 - \xi_i \quad \forall i \\
\xi_i & \geq 0, \sum_{j=1}^n \xi_j \leq \nu
\end{align*}
\]
SVM as penalization method

- Original quadratic program with linear constraints:
  \[
  \min_{w,b} \|w\|_2^2 \\
  y_i(x_i^Tw + b) \geq 1 - \xi_i \quad \forall i \\
  \xi_i \geq 0, \sum_{j=1}^n \xi_j \leq \nu
  \]

- Using same constrained convex optimization trick as for lasso:
  For any \( \nu \geq 0 \) there exists a \( \lambda \geq 0 \) such that the solution the following solution is equivalent:
  \[
  \sum_{i=1}^n \max\{0, 1 - y_i(b + x_i^Tw)\} + \lambda \|w\|_2^2
  \]
Machine Learning Problems

- Have a bunch of iid data of the form:
  \[ \{(x_i, y_i)\}_{i=1}^{n} \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R} \]

- Learning a model’s parameters:
  Each \( \ell_i(w) \) is convex.

  \[
  \sum_{i=1}^{n} \ell_i(w)
  \]

Hinge Loss: \( \ell_i(w) = \max\{0, 1 - y_i x_i^T w\} \)

Logistic Loss: \( \ell_i(w) = \log(1 + \exp(-y_i x_i^T w)) \)

Squared error Loss: \( \ell_i(w) = (y_i - x_i^T w)^2 \)

How do we solve for \( w \)? The last two lectures!
Perceptron is optimizing what?

Perceptron update rule:

\[
\begin{bmatrix}
w_{k+1} \\
b_{k+1}
\end{bmatrix} = \begin{bmatrix}
w_k \\
b_k
\end{bmatrix} + y_k \begin{bmatrix}
x_k \\
1
\end{bmatrix} 1\{y_i(b + x_i^T w) < 0\}
\]

SVM objective:

\[
\sum_{i=1}^{n} \max\{0, 1 - y_i(b + x_i^T w)\} + \lambda ||w||_2^2 = \sum_{i=1}^{n} \ell_i(w, b)
\]

\[
\nabla_w \ell_i(w, b) = \begin{cases} 
-x_i y_i + \frac{2\lambda}{n} w & \text{if } y_i(b + x_i^T w) < 1 \\
0 & \text{otherwise}
\end{cases}
\]

\[
\nabla_b \ell_i(w, b) = \begin{cases} 
-y_i & \text{if } y_i(b + x_i^T w) < 1 \\
0 & \text{otherwise}
\end{cases}
\]

Perceptron is just SGD on SVM with \(\lambda = 0, \eta = 1\)!
SVMs vs logistic regression

- We often want probabilities/confidences, logistic wins here?
SVMs vs logistic regression

- We often want probabilities/confidences, logistic wins here?
- No! Perform isotonic regression or non-parametric bootstrap for probability calibration. Predictor gives some score, how do we transform that score to a probability?
SVMs vs logistic regression

- We often want probabilities/confidences, logistic wins here?
- No! Perform isotonic regression or non-parametric bootstrap for probability calibration. Predictor gives some score, how do we transform that score to a probability?

- For classification loss, logistic and svm are comparable
- Multiclass setting:
  - Softmax naturally generalizes logistic regression
  - SVMs have
- What about good old least squares?
What about multiple classes?