SVAN 2016 Mini Course: Stochastic Convex Optimization Methods in Machine Learning

Mark Schmidt
University of British Columbia, May 2016
www.cs.ubc.ca/~schmidtm/SVAN16

Some images from this lecture are taken from Google Image Search.
Last Time: Supervised Learning

• We discussed **supervised learning**:  
  – We have a set of inputs $x_i$ and a corresponding output $y_i$.  
  – Food allergy example:  
    • $x_i$ is the quantities of food we ate on day ‘i’.  
    • $y_i$ is the level of IgE we measure on day ‘i’.  
  – The goal is to learn a function ‘f’ such that $(f(x_i) - y_i)$ is small.  

• We introduced standard notation for supervised learning:
Last Time: Linear Regression and Least Squares

• We considered the special case of linear regression:

\[ \hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \cdots + w_d x_{id} = W^T x_i \]

• To fit this model, a classic approach is least squares:

\[
\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} (w^T x_i - y_i)^2
\]

• Which we can write in matrix notation as:

\[
\min_{w \in \mathbb{R}^d} \| Xw - y \|_2^2 \\
\text{Solution: } w = (X^T X)^{-1} X^T y \text{ (invertible } X^T X)\]
Last Time: Nonlinear Basis

- Change of basis allows nonlinear functions with linear regression:

\[
X = \begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{bmatrix}
\]

\[
X_{poly} = \begin{bmatrix}
  1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\
  1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & x_n & (x_n)^2 & \cdots & (x_n)^p
\end{bmatrix}
\]
Fundamental Trade-Off of Machine Learning

- Same trade-off exists as we add more features:
  - More features means lower training error.
  - More features means training error is worse approximation of test error.
Controlling Complexity

• We know that complex models can overfit.
• But usually the “true” mapping from $x_i$ to $y_i$ is complex.
• So what do we do???

• There are many possible answers:
  – Model averaging: average over multiple models to decrease variance.
  – Regularization: add a penalty on the complexity of the model.
L2-Regularization

• Our standard least squares formulation:

\[ \arg\min_{w \in \mathbb{R}^d} \frac{1}{2} ||Xw - y||^2 \]

• Standard regularization strategy is to add a penalty on the L2-norm:

\[ \arg\min_{w \in \mathbb{R}^d} \frac{1}{2} ||Xw - y||^2 + \frac{\lambda}{2} ||w||^2 \]

• Regularization parameter \( \lambda \) controls ‘strength’ of regularization:
  – If \( \lambda \) is large then it forces ‘\( w \)’ to be very small: low complexity.
  – If \( \lambda \) is tiny then ‘\( w \)’ can be get huge: high complexity.

• Has been re-invented several times:
  – Tikhonov regularization, ridge regression, etc.
L2-Regularization

• In terms of fundamental trade-off:
  – Regularization increases training error.
  – Regularization makes training error a better approximation of test error.

• How should you choose \( \lambda \)?
  – Theory: as ‘n’ grows \( \lambda \) should be in the range \( O(1) \) to \( O(n^{-1/2}) \).
  – Practice: optimize validation set or cross-validation error.
    • This almost always decreases the test error.

• How do you compute ‘w’?
Ridge Regression Calculation

Objective: \( f(w) = \frac{1}{2} (y - Xw)^T (y - Xw) + \frac{\lambda}{2} w^T w \).

Gradient: \( \nabla f(w) = X^T Xw - X^T y + \lambda w \)

Setting to zero: \( X^T Xw + \lambda w = X^T y \) or \( (X^T X + \lambda I)w = X^T y \).

Pre-multiply by \( (X^T X + \lambda I)^{-1} \), which always exists:

In Matlab:

\[ w = (X^T X + \lambda I)^{-1} X^T y. \]
Why use L2-Regularization?

• Mark says: “You should always use regularization.”
• “Almost always improves test error” should already convince you.
• But here are more reasons:
  1. Solution ‘w’ is unique.
  2. Does not require $X'X$ to be invertible.
  3. Solution ‘w’ is less sensitive to changes in $X$ or $y$.
  4. You can use Cholesky factorization instead of LU factorization.
  5. Makes large-scale methods for computing ‘w’ run faster.
  6. Stein’s paradox: if $d \geq 3$, regularization moves us closer to ‘true’ $w$.
  7. In the worst case you just set $\lambda$ small and get the same performance.
(pause)
Parametric vs. Non-Parametric

• Polynomials are not the only possible bases:
  – Common to use exponentials, logarithms, trigonometric functions, etc.
  – The right basis will vastly improve performance.
  – But when you have a lot of features, the right basis may not be obvious.

• The above bases are parametric model:
  – The size of the model does not depend on the number of training examples ‘n’.
  – As ‘n’ increases, you can estimate the model more accurately.
  – But at some point, more data doesn’t help because model is too simple.

• Alternative is non-parametric models:
  – Size of the model grows with the number of training examples.
  – Model gets more complicated as you get more data.
  – You can model very complicated functions where you don’t know the right basis.
Non-Parametric Basis: RBFs

• Radial basis functions (RBFs):
  – Non-parametric bases that depend on distances to training points.
• Most common example is Gaussian or squared exponential:

\[ f(x) = \exp\left( -\frac{\|x - x_i\|^2}{2\sigma^2} \right) \]
Non-Parametric Basis: RBFs

- Radial basis functions (RBFs):
  - Non-parametric bases that depend on distances to training points.
  - Most common example is Gaussian or squared exponential:

\[
X_{rbf} = \begin{bmatrix}
    e^{\frac{-\|x_1-x_1\|^2}{2\sigma^2}} & e^{\frac{-\|x_1-x_2\|^2}{2\sigma^2}} & \cdots & e^{\frac{-\|x_1-x_n\|^2}{2\sigma^2}} \\
    e^{\frac{-\|x_2-x_1\|^2}{2\sigma^2}} & e^{\frac{-\|x_2-x_2\|^2}{2\sigma^2}} & \cdots & e^{\frac{-\|x_2-x_n\|^2}{2\sigma^2}} \\
    \vdots & \vdots & \ddots & \vdots \\
    e^{\frac{-\|x_n-x_1\|^2}{2\sigma^2}} & e^{\frac{-\|x_n-x_2\|^2}{2\sigma^2}} & \cdots & e^{\frac{-\|x_n-x_n\|^2}{2\sigma^2}} 
\end{bmatrix}
\]

\[
\hat{y}_j = w_1 \begin{tikzpicture}
    \draw[fill=gray!20] (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
\end{tikzpicture} + w_2 \begin{tikzpicture}
    \draw[fill=gray!20] (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
\end{tikzpicture} + \cdots + w_k \begin{tikzpicture}
    \draw[fill=gray!20] (0,0) -- (1,0) -- (1,1) -- (0,1) -- cycle;
\end{tikzpicture}
\]

- Gaussian RBFs are universal approximators (compact subets of \( \mathbb{R}^d \))
  - Can approximate any continuous function to arbitrary precision.
Non-Parametric Basis: RBFs

- RBF basis for different values of $\sigma$:

\[ X = \begin{bmatrix} 1 & -x_1 & -x_2 & \cdots & -x_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & -x_1 & -x_2 & \cdots & -x_n \end{bmatrix} X_{\text{rbf}} \]

Could add bias and linear basis:

Now it defaults to linear regression instead of 0 away from data.
RBFs, Regularization, and Validation

• Very effective model:
  – RBF basis with L2-regularization and cross-validation to choose $\sigma$ and $\lambda$.
  – Expensive at test time: need distance to all training examples.
RBFs, Regularization, and Validation

• RBF basis with L2-regularization for different values of $\sigma$ and $\lambda$.

• At least one of these models is often a good fit.
(pause)
Alternatives to Squared Error

• Squared error is **computationally convenient** choice:
  – Solution involves solving a linear system.
  \[ w = (X^T X + \lambda I)^{-1} X^T y \]

• But it’s usually **not the right choice**:
  – Corresponds to assuming error are normally distributed (later in lecture).
  – Makes it **sensitive to outliers** or large errors.
  – Makes it **inappropriate with restrictions** on \( y \) (like binary or censored).

• There are many **alternatives to squared error**:
  – But these have computational implications.
Least Squares with Outliers

• Consider fitting least squares with an outlier in the labels:
  – Observation that is unusually different from the others.

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Egg</td>
<td>Milk</td>
<td>Fish</td>
<td>Wheat</td>
</tr>
<tr>
<td>0</td>
<td>0.7</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>1.2</td>
<td>0</td>
</tr>
</tbody>
</table>

• Some sources of outliers:
  – Errors, contamination of data from different distribution, rare events.
Least Squares with Outliers

• Consider fitting least squares with an outlier in the labels:

\[ x \leftarrow \text{"outlier"}; \text{it's not like the others.} \]

This is probably what we want.
Least Squares with Outliers

• Consider fitting least squares with an outlier in the labels:

\[ x \leftarrow \text{"outlier"}; \text{it's not like the others.} \]

• Least squares is very sensitive to outliers.
Least Squares with Outliers

• Squaring error shrinks small errors, and **magnifies large errors**:

\[
\begin{align*}
\text{Absolute errors} & \quad \text{Square errors} \\
(5)^2 &= 25 \\
\left(\frac{1}{2}\right)^2 &= \frac{1}{4}
\end{align*}
\]

• Outliers (large error) influence ‘w’ much more than other points.
Least Squares with Outliers

• Squaring error shrinks small errors, and magnifies large errors:

  Absolute Errors:  Squared Errors:

  ![Graph showing absolute and squared errors]

• Outliers (large error) influence ‘w’ much more than other points.
  – Good if outlier means ‘plane crashes’, bad if it means ‘data entry error’.
Robust Regression

- Robust regression objectives put less focus on far-away points.
- For example, just use absolute error:

\[
\text{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^{n} |w^T x_i - y_i|
\]

- Now decreasing ‘small’ and ‘large’ errors is equally important.
- In matrix notation, we can write this as minimizing \( L1 \)-norm:

\[
\sum_{i=1}^{n} |r_i| = \sum_{i=1}^{n} |w^T x_i - y_i|
\]

Let “residual” vector \( r \) have elements

\[
r_i = w^T x_i - y_i
\]

so \( r = Xw - y \)
Squared Error vs. Absolute Error

• Comparing squared error absolute error:

\[
\text{Absolute Errors:} \quad \text{Squared Errors:}
\]

\[
\underset{w \in \mathbb{R}^d}{\text{argmin}} \frac{1}{2} \| y - Xw \|_2^2
\]
Squared Error vs. Absolute Error

- Comparing squared error absolute error:

\[ \arg\min_{w \in \mathbb{R}^d} \|y - xw\|_1 \]
Regression with the L1-Norm

- Unfortunately, minimizing the absolute error is harder:
  - Gradient doesn’t always exist.
  - Generally, harder to minimize non-smooth than smooth functions.
  - But we can formulate minimize absolute error as a linear program.
Converting into Constrained Problems

• Key observation:
  – Absolute value is **maximum of smooth functions**: \( |w| = \max \sum w_j - w^3 \)

• We can convert to minimizing **smooth function with constraints**:
  1. Replace **maximum** with new variable, constrained to upper-bound max.
  2. Replace individual constraint with **constraint for each element** of max.

\[
\begin{align*}
\arg\min_{w \in \mathbb{R}} |w| \quad &\iff \quad \arg\min_{w \in \mathbb{R}} \max_{j} w_j - w^3 \quad \iff \quad \arg\min_{w \in \mathbb{R}, r \in \mathbb{R}} r_j \text{ s.t. } r \geq \max_{j} w_j - w^3 \\
&\quad \uparrow \\
&\quad \text{We must have } r = |w| \text{ at solution, otherwise constraints are violated or we could decrease } r.
\end{align*}
\]
Minimizing Absolute Error as Linear Program

• We can apply the same steps to a sum of max functions:

\[
\begin{align*}
\arg\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} |\mathbf{w}^T \mathbf{x}_i - y_i| & \iff \arg\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} \max\left\{ \mathbf{w}^T \mathbf{x}_i - y_i, y_i - \mathbf{w}^T \mathbf{x}_i \right\} \\
& \iff \arg\min_{\mathbf{w} \in \mathbb{R}^d, \mathbf{r} \in \mathbb{R}^n} \sum_{i=1}^{n} r_i \quad \text{subject to } r_i \geq \max\left\{ \mathbf{w}^T \mathbf{x}_i - y_i, y_i - \mathbf{w}^T \mathbf{x}_i \right\} \\
& \quad \text{for all } i, \\
& \iff \arg\min_{\mathbf{w} \in \mathbb{R}^d, \mathbf{r} \in \mathbb{R}^n} \sum_{i=1}^{n} r_i \quad \text{subject to } r_i \geq \mathbf{w}^T \mathbf{x}_i - y_i \quad \text{and} \quad r_i \geq y_i - \mathbf{w}^T \mathbf{x}_i \\
& \quad \text{for all } i.
\end{align*}
\]

• This is a linear program:

– Minimizing a linear function subject to linear constraints.
– We can efficiently solve ‘medium-sized’ linear programs: Matlab’s ‘linprog’.
– There are other linear program formulations of this problems.
‘Brittle’ Regression

• What if you really care about getting the outliers right?
  – You want best performance on worst training example.
  – For example, if in worst case the plane can crash.

• In this case you can use something like the infinity-norm:

\[
\arg \min_{x \in \mathbb{R}^n} \|x_n - y\|_{\infty} \quad \quad \|z\|_{\infty} = \max_i \{ |z_i| \}
\]

• Very sensitive to outliers (brittle), but worst case will be better.
Robust vs. Brittle Regression

- We said that squared error is **sensitive to outliers**:
  - **Absolute error** is less sensitive: can be solved as a linear program.
  - **Maximum error** is more sensitive: can also be solved as linear program.

\[
\begin{align*}
\text{minimizes } & \quad \sum (w^T x_i - y_i)^2 \\
\text{minimizes } & \quad \max_i \sum |w^T x_i - y_i| \\
\text{minimizes } & \quad \sum |w^T x_i - y_i|
\end{align*}
\]
Very Robust Regression?

• Can we be more robust?

  - Very robust: eventually “gives up” on large errors.
  - But finding optimal ‘w’ is NP-hard.
    - Absolute value is the most robust that is not NP-hard.
The ‘Best’ Machine Learning Model

• What is the ‘best’ machine learning model?
  – SVMs? Random forests? Deep learning?

• No free lunch theorem:
  – There is no ‘best’ model that achieves the best test error for every problem.
  – If model A works better than model B on one dataset, there is another dataset where model B works better.

• Asking what is the ‘best’ machine learning model is like asking which is ‘best’ among “rock”, “paper”, and “scissors”.

• Caveat of no free lunch (NFL) theorem:
  – The world is very structured, some datasets are more likely than others.
  – Model A could be better than model B on a huge variety of practical applications.

• Machine learning emphasizes models useful across applications.
Last Time: Robust Regression

• We said that squared error is **sensitive to outliers**:  
  – Absolute error is less sensitive: can be solved as a linear program.
Motivation: Identifying Important E-mails

• We have a big collection of e-mails:
  – Marked as ‘important’ if user took some action based on them.

• We want to write a program that identifies ‘important’ e-mails?
• Can we formulate as supervised learning?
Supervised Learning Representation for E-mails

• For e-mail ‘i’, the target label $y_i$ is binary:
  – +1: “e-mail is important”.
  – -1: “e-mail is not important”.
  – Classification: supervised learning with discrete labels.

• What are the right features $x_i$ (basis) for e-mails?
  – Use bag of words:
    • “CPSC”, “Expedia”, “vicodin”.
    • Binary “Expedia” feature is 1 if phrase “Expedia” is in the message, and 0 otherwise.
  – Could add phrases:
    • “you’re a winner”, “CPSC 540”.
  – Could add regular expressions:
    • <recipient name>, <sender domain == “mail.com”>
Can we make **personalized** predictions?

- Some messages ‘universally’ important:
  - “This is your mother, something terrible happened, give me a call ASAP.”
- Some messages may be important to one user but not others.
The Big Global/Local Feature Table

"local" features for user 1: these are non-zero for user 1, 0 for other users

"global" features: all users share these.

"local" features for user 2.
Predicting Importance of E-mail For New User

• Consider a new user:
  – Start out with no information about them.
  – Use global features to predict what is important to generic user.

\[ \hat{y}_i = \text{sign}(w_g^T x_g) \]

• With more data, update global features and user’s local features:
  – Local features make prediction personalized.

\[ \hat{y}_i = \text{sign}(w_g^T x_g + w_u^T x_u) \]

• G-mails system: classification with logistic regression.
Classification Using Regression?

• Usual approach to do binary classification with regression:
  – Code $y_i$ as ‘+1’ for one class and ‘-1’ for the other class.

• Fit a linear regression model:
  \[ f_{\hat{w}} = \sum_{j=1}^{d} w_j x_{i,j} \]
  \[ = w^T \chi_i \]

• Classify by take the sign (i.e., closer ‘-1’ or ‘+1’?):
  \[ \hat{y}_i = \text{sign}(w^T \chi_i) \].
Classification using Regression

The figure illustrates a linear regression model. The equation $w^T x_i = 0$ represents the decision boundary. Points to the left of the boundary are classified as -1, and points to the right are classified as +1. The text highlights the importance of features in the classification process.
Classification using Regression

• Can use our tricks (e.g., RBF basis, regularization) for classification.
• But, usual error functions do weird things:
Classification Using Regression

- What went wrong?
  - “Good” errors vs. “bad” errors.
Classification Using Regression

- What went wrong?
  - “Good” errors vs. “bad” errors.

The bad errors of the perfect classifier are HUGE.

# times we see 'vicodin'
Comparing Loss Functions

\[(y_i - \hat{y}_i)^2\]

big penalty for being too right.

\[^\hat{y}_i = -1\]

"loss" for predicting $\hat{y}_i$ when true label is $y_i = -1$.

"bad" error: you should not penalize for putting $\hat{y}_i$ here.

"good" error: putting $\hat{y}_i$ here is bad.
Comparing Loss Functions

\((y_i - \hat{y}_i)^2\) which gives a big penalty for being too right.

\(L_1\) does not fix this issue.

A "bad" error: you should not penalize for putting \(\hat{y}_i\) here.

"good" error: putting \(\hat{y}_i\) here is bad.

"loss\) for predicting \(\hat{y}_i\) when true label is \(y_i = -1\).

Prediction \(\hat{y}_i\).
Comparing Loss Functions

L_1 does not fix this issue.

"bad" error: you should not penalize for putting \( \hat{y}_i \) here.

"good" error: putting \( \hat{y}_i \) here is bad.

\[ (y_i - \hat{y}_i)^2 \]

big penalty for being too right.

\( y_i = -1 \)

"loss" for predicting \( \hat{y}_i \) when true label is \( y_i = -1 \).

What we want: "0-1" loss, number of errors.
0-1 Loss Function and Tractable Approximations

• The **0-1 loss function** is the number of errors after taking the sign.
  – If a perfect classifier exists, you can find one as a linear program.
  – Otherwise, it’s **NP-hard** to minimize 0-1 loss:
    • We do not expect that efficient algorithms exist.

• Tractable alternatives to 0-1 loss:
  – **Hinge loss**: upper-bound on 0-1 loss that can be written as linear program.
  – **Logistic loss**: differentiable function similar to hinge loss.
0-1 Loss Function and Tractable Approximations

"hinge" loss:

"loss" for predicting $\hat{y}_i$ when true label is $y_i = -1$.

0-1 loss.

prediction $\hat{y}_i$
0-1 Loss Function and Tractable Approximations

"hinge" loss:

"logistic" loss:

0-1 loss.

\( y_i = -1 \)

\( \hat{y}_i \)

"loss" for predicting \( \hat{y}_i \) when true label is \( y_i = -1 \).
Hinge Loss and Support Vector Machines

• **Hinge loss** is given by:
  \[
  \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max \{ 0, 1 - y_i w^T x_i \}
  \]
  – Can be written as a **linear program** using our max trick.
  – Solution will be a perfect classifier, if one exists.

• **Support vector machine (SVM)** is hinge loss with L2-regularization.
  \[
  \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max \{ 0, 1 - y_i w^T x_i \} + \frac{\lambda}{2} \|w\|^2
  \]
  – Can be written as a **quadratic program** using our max trick
    • Quadratic objective with linear constraints.
  – Solution will be perfect classifier, if one exists and \( \lambda \) is small enough.
  – **Maximizes margin**: maximizes distance of data to decision boundary.
Logistic Regression

• **Logistic regression** minimizes logistic loss:

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))
\]

• You can/should also add regularization:

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \frac{\lambda}{2} \|w\|^2
\]

• These can’t be written as linear/quadratic programs:
  – But they’re **differentiable**: we’ll discuss how to solve them next time.
Logistic Regression and SVMs

• SVMs and logistic regression are used EVERYWHERE!
• Why?
  – Training and testing are both fast, even for “large-scale” problems.
  – It is easy to understand what the weights ‘$w_j$’ mean.
  – With high-dimensional features and regularization, often good test error.
  – Otherwise, often good test error with RBF basis and regularization.
  – For logistic regression, predictions have probabilistic interpretation.

\[
\text{If } \Pr(y_i = +1 | w, x_i) = \sigma(w^T x_i) \text{ then minimizing logistic loss corresponds to maximum likelihood estimate.}
\]
Discussion: Probabilistic Interpretation

- Why is probabilistic interpretation important?
  - We can return a probabilistic prediction:
    
    Instead of \( \hat{y}_i = 1 \), say that \( p(\hat{y}_i = 1 | w_j x_i) = 99\% \) or \( p(\hat{y}_i = 1 | w_j x_i) = 51\% \)
  
  - For complicated \( y_i \), it may be easier to define probability than loss.
  
  - We can talk about maximizing utility:

<table>
<thead>
<tr>
<th>Predict / True</th>
<th>True ‘spam’</th>
<th>True ‘not spam’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict ‘spam’</td>
<td>TP: 0</td>
<td>FP: 100</td>
</tr>
<tr>
<td>Predict ‘not spam’</td>
<td>FN: 10</td>
<td>TN: 0</td>
</tr>
</tbody>
</table>

\[
E \left[ C(\hat{y}_i = \text{spam}\mid x_i) \right] = p(y_i = \text{spam} \mid x_i) C(\hat{y}_i = \text{spam}, y_i = \text{spam}) \\
+ p(y_i = \text{not spam} \mid x_i) C(\hat{y}_i = \text{spam}, y_i = \text{not spam})
\]

Predict "not spam" even if \( \hat{y}_i = "spam" \) if expected cost of "not spam" is lower.
Maximum Likelihood and MAP Estimation

• Unregularized logistic regression is **maximum likelihood** solution:
  – Maximize likelihood of data given model parameters.
  – Problem with maximum likelihood:
    • data could be very likely in some **very unlikely model** from family.
    • E.g., complex model overfits by memorizing the data.

• Regularized logistic regression is **MAP** (maximum a posteriori):

\[
\arg\max_{w \in \mathbb{R}^d} p(w | y, X) \iff \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} - \log p(y_i | w, x_i) - \log p(w)
\]

  – Model is a random variable, and we need to find most likely model.
  – Can take into account that complex models are likely to overfit.
Multi-Class Logistic Regression

• Supposed $y_i$ takes values from an unordered discrete set of classes.

$X_i \in \mathbb{R}^d \quad y_i \in \{1, 2, 3, 4, \ldots, K\}$

• Standard model:
  – Use a ‘$d$’-dimensional weight vector $w_c$ for each class ‘$c$’.
  – Try to make inner-product $w_c^T x_i$ big when ‘$c$’ is the true label ‘$y_i$’.
  – Classify by finding largest inner-product:

$$\hat{y_i} = \arg\max_c \{ w_c^T x_i \}$$

(Also exist models for ordered classes or count data)
Multi-Class Logistic Regression

We have a parameter matrix \( W = \left\{ \begin{array}{c} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_k \end{array} \right\} \) to make a prediction. Compute \( W^T x_i \) and compute maximum.

We want a loss function that will make \( w_c^T x_i \) big when \( c \) is the true label \( y_i \) and will otherwise make \( w_c^T x_i \) small.

We can define probability using softmax function:

\[
p(y_i = c | W, x_i) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{K} \exp(w_{c'}^T x_i)}
\]

To fit model, use

\[
-\log p(y_i | W, x_i) = -w_{y_i}^T x_i + \log \left( \sum_{c'=1}^{K} \exp(w_{c'}^T x_i) \right)
\]

If \( k=3 \):

\[
p(y_i | W, x_i) = \frac{\exp(w_{y_i}^T x_i)}{\exp(w_{y_1}^T x_i) + \exp(w_{y_2}^T x_i) + \exp(w_{y_3}^T x_i)}
\]

Generalizes sigmoid:

- special case of \( k=2 \) and \( w_2 = 0 \)
Course Roadmap

• Part 1: Overview of Machine Learning
  – **Linear models**: change of basis, regularization, loss functions.
  – **Basics of learning theory**: Training vs. test error, bias-variance, fundamental trade-off, no free lunch.
  – **Probabilistic learning principles**: Maximum likelihood, MAP estimation, loss functions.

• Part 2: **Large-scale machine learning**.
  – Why are SVMs/logistic easy while minimizing number of errors is hard?
  – How do we fit these models to **huge datasets**?
Summary

- **Regularization**: allows complicated models by penalizing complexity.
- **Radial basis functions**: non-parametric universal basis.
- **Robust regression models**: more suitable when we have outliers.
- **Converting non-smooth problems to constrained smooth problems**.
- **No free lunch**: there is no ‘best’ machine learning model.
- **SVMs and logistic regression**: more suitable losses for classification.
- **MLE and MAP**: probabilistic interpretation to losses/regularizers.
- **Softmax loss** to model discrete $y_i$, other losses can be derived.

— Next time: Why is logistic easy while minimizing number of errors is hard?