Announcements

• Quiz 4 is due **Monday, October 10 at 8pm**
  • Quiz 5 posted Today evening

• **Project Teams Update**
  • Most likely allowing teams of 2 (will confirm shortly)
  • *Let us know if you prefer to split up and be reassigned*
  • Similar expectations on novel contributions but proportionally less work

• **HW 3 Posted**
  • Due Wednesday, October 19 (2 weeks from today), lease start early!
  • *Also, HW 2 late deadline is tonight at 8pm!*
Application of KNN

Recap: Ensembles

- Meta-algorithms for combining models to improve their performance

- For an ensemble learning algorithm, two design decisions:
  - How to learn base models?
  - How to combine learned base models?
Recap: Ensemble Design Decisions

• How to learn the base models $f_1(x), \ldots, f_k(x)$?
  • Intuition: Need diversity
  • Handcrafted models
  • Bagging: Subsample examples and/or features
  • Boosting: Iteratively upweight currently incorrect examples

• How to combine the learned base models?
  • Average or majority vote
  • Learn a model $g_\beta(f_1(x), \ldots, f_k(x))$ treating $f_1(x), \ldots, f_k(x)$ as “features”
Recap: Ensembles of Decision Trees
Recap: Random Forests

- **Ensemble strategy**
  - Bagging applied to unpruned decision trees
  - Randomly subsample $\sqrt{d}$ features at each split
  - Average random trees

- **Intuition**
  - Unpruned decision trees have high variance
  - Randomness enables us to “average away” excess variance
  - Cannot “overfit” by using too many trees
Recap: Boosting

• **Ensemble strategy**
  • Train depth-limited decision tree on weighted dataset
  • Iteratively upweight incorrectly classified examples

• **Intuition**
  • Depth-limited decision trees have high bias
  • Learning many models increases variance
  • Can overfit by learning too many trees (but often does not in practice)
AdaBoost (Freund & Schapire 1997)

• **Input**
  • Training dataset $Z$
  • Learning algorithm $\text{Train}(Z, w)$ that can handle weights $w$
  • Hyperparameter $T$ indicating number of models to train

• **Output**
  • Ensemble of models $F(x) = \sum_{t=1}^{T} \beta_t \cdot f_t(x)$
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. for $t \in \{1, \ldots, T\}$
3. $f_t \leftarrow \text{Train}(Z, w_t)$
4. $\epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. return $F(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$
AdaBoost

1. \( w_1 \leftarrow \left( \frac{1}{n}, ..., \frac{1}{n} \right) \) (\( w_{1,i} \) weight for \( (x_i, y_i) \))
2. for \( t \in \{1, ..., T\} \)
3. \( f_t \leftarrow \text{Train}(Z, w_t) \)
4. \( \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t) \)
5. \( \beta_t \leftarrow \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right) \)
6. \( w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)} \) (for all \( i \))
7. return \( F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x)) \)
AdaBoost

1. \( w_1 \leftarrow \left( \frac{1}{n}, ..., \frac{1}{n} \right) \) (\( w_{1,i} \) weight for \( (x_i, y_i) \))
2. \( \text{for } t \in \{1, ..., T\} \)
3. \( f_t \leftarrow \text{Train}(Z, w_t) \)
4. \( \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t) \)
5. \( \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} \)
6. \( w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot \epsilon_t} \) (for all \( i \))
7. \( \text{return } F(x) = \text{sign}(\sum_{t=0}^{T} \beta_t \cdot f_t(x)) \)

\( \beta_t \) becomes larger as \( \epsilon_t \) becomes smaller.
AdaBoost

1. $w_1 \leftarrow \left( \frac{1}{n}, ..., \frac{1}{n} \right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. for $t \in \{1, ..., T\}$
3. $f_t \leftarrow \text{Train}(Z, w_t)$
4. $\epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. return $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

Use convention $y_i \in \{-1, +1\}$
If correct ($y_i = f_t(x_i)$) then multiply by $e^{-\beta_t}$
If incorrect ($y_i \neq f_t(x_i)$) then multiply by $e^{\beta_t}$
AdaBoost

1. \( w_1 \leftarrow \left( \frac{1}{n}, ..., \frac{1}{n} \right) \) (\( w_{1,i} \) weight for \( (x_i, y_i) \))

2. \textbf{for} \( t \in \{1, ..., T\} \)

3. \( f_t \leftarrow \text{Train}(Z, w_t) \)

4. \( \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t) \)

5. \( \beta_t \leftarrow \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t} \)

6. \( w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)} \) (for all \( i \))

7. \textbf{return} \( F(x) = \text{sign} \left( \sum_{t=1}^{T} \beta_t \cdot f_t(x) \right) \)
AdaBoost

1. \( w_1 \leftarrow \left( \frac{1}{n}, ..., \frac{1}{n} \right) \) (weight for \((x_i, y_i))\)
2. \( \text{for } t \in \{1, ..., T\} \)
3. \( f_t \leftarrow \text{Train}(Z, w_t) \)
4. \( \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t) \)
5. \( \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} \)
6. \( w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)} \) (for all \(i\))
7. \( \text{return } F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x)) \)
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, ..., \frac{1}{n}\right) \ (w_{1,i} \text{ weight for } (x_i, y_i))$

2. for $t \in \{1, ..., T\}$

3. $f_t \leftarrow \text{Train}(Z, w_t)$

4. $\epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$

5. $\beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

6. $w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i f_t(x_i)}$ (for all $i$)

7. return $F(x) = \text{sign} \left( \sum_{t=1}^{T} \beta_t \cdot f_t(x) \right)$
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, ..., \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. for $t \in \{1, ..., T\}$
3. $f_t \leftarrow \text{Train}(Z, w_t)$
4. $\epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t y_i f_t(x_i)}$ (for all $i$)
7. return $F(x) = \text{sign} \left( \sum_{t=1}^{T} \beta_t \cdot f_t(x) \right)$
AdaBoost

1. \( w_1 \leftarrow \left( \frac{1}{n}, ..., \frac{1}{n} \right) \) (\( w_{1,i} \) weight for \( (x_i, y_i) \))
2. \( \text{for } t \in \{1, ..., T\} \)
3. \( f_t \leftarrow \text{Train}(Z, w_t) \)
4. \( \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t) \)
5. \( \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} \)
6. \( w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)} \) (for all \( i \))
7. return \( F(x) = \text{sign} \left( \sum_{t=1}^{T} \beta_t \cdot f_t(x) \right) \)
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right) (w_{1,i} \text{ weight for } (x_i, y_i))$
2. for $t \in \{1, \ldots, T\}$
3. $f_t \leftarrow \text{Train}(Z, w_t)$
4. $\epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)} \text{ (for all } i)$
7. return $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

final model is average of base models weighted by their performance
AdaBoost Summary

• **Strengths:**
  • Fast and simple to implement
  • No hyperparameters (except for $T$, which is robust in practice)
  • Very few assumptions on base models (except they should be low capacity)

• **Weaknesses:**
  • Can perform poorly when there is insufficient data
  • No way to parallelize
  • **Specific to classification!**
Boosting as Gradient Descent

• Both algorithms: new model = old model + update

• Gradient Descent:

\[ \theta_{t+1} = \theta_t - \alpha \cdot \nabla_{\theta} L(\theta_t; Z) \]

• Boosting:

\[ F_{t+1}(x) = F_t(x) + \beta_{t+1} \cdot f_{t+1}(x) \]

• Here, \( F_t(x) = \sum_{i=1}^{t} \beta_i \cdot f_i(x) \)
Boosting as Gradient Descent

• Assuming $\beta_t = 1$ for all $t$, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i)$$
Boosting as Gradient Descent

• Assuming $\beta_t = 1$ for all $t$, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i) \approx y_i$$

• Rewriting this equation, we have

$$f_{t+1}(x_i) = F_{t+1}(x_i) - F_t(x_i) \approx y_i - F_t(x_i)$$

“residuals”, i.e., error of the current model
Boosting as Gradient Descent

• In other words, at each step, boosting is training the next model $f_{t+1}$ to approximate the residual:

$$f_{t+1}(x_i) \approx y_i - F_t(x_i)$$

“residuals”, i.e., error of the current model

• **Idea:** Train $f_{t+1}$ directly to predict residuals $y_i - F_t(x_i)$

• **This strategy works for regression as well!**
Boosting as Gradient Descent

**Algorithm:** For each $t \in \{1, \ldots, T\}$:

- **Step 1:** Train $f_{t+1}$ using dataset

  $$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$$

- **Step 2:** Take

  $$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model $F_T$
Boosting as Gradient Descent

• Consider losses of the form

\[
L(F; Z) = \frac{1}{n} \sum_{i=1}^{n} \tilde{L}(F(x_i); y_i)
\]

• In other words, sum of individual label-level losses \( \tilde{L}(\hat{y}; y) \) of a prediction \( \hat{y} = F(x) \) if the ground truth label is \( y \)

• For example, \( \tilde{L}(\hat{y}; y) = \frac{1}{2} (y - \hat{y})^2 \) yields the MSE loss
Boosting as Gradient Descent

• Residuals are the gradient of the squared error $\tilde{L}(y, \hat{y}) = \frac{1}{2} (y - \hat{y})^2$:

$$-\frac{\partial \tilde{L}}{\partial \hat{y}} (F_t(x_i); y_i) = y_i - F_t(x_i) = \text{residual}_i$$

• For general $\tilde{L}$, instead of $\{(x_i, y_i - F_t(x_i))\}_{i=1}^n$ we can train $f_{t+1}$ on $Z_{t+1} = \left\{ \left( x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}} (F_t(x_i); y_i) \right) \right\}_{i=1}^n$
Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, ..., T\}$:
  - **Step 1:** Train $f_{t+1}$ using dataset

  $$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^n$$

  - **Step 2:** Take

  $$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model $F_T$
Boosting as Gradient Descent

• **Algorithm:** For each $t \in \{1,\ldots,T\}$:
  
  • **Step 1:** Train $f_{t+1}$ using dataset

  $$Z_{t+1} = \left\{ \left( x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i) \right) \right\}_{i=1}^{n}$$

  • **Step 2:** Take

  $$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

• Return the final model $F_T$
Boosting as Gradient Descent

• Casts ensemble learning in the **loss minimization framework**
  - **Model family**: Sum of base models $F_T(x) = \sum_{t=1}^{T} f_t(x)$
  - **Loss**: Any differentiable loss expressed as

\[
L(F; Z) = \sum_{i=1}^{n} \tilde{L}(F(x_i), y_i)
\]

• Gradient boosting is a general paradigm for training ensembles with specialized losses (e.g., most NLL losses)
Gradient Boosting in Practice

- Gradient boosting with depth-limited decision trees (e.g., depth 3) is one of the most powerful off-the-shelf classifiers available
  - **Caveat:** Inherits decision tree hyperparameters

- XGBoost is a very efficient implementation suitable for production use
  - A popular library for gradient boosted decision trees
  - Optimized for computational efficiency of training and testing
  - Used in many competition winning entries, across many domains
  - [https://xgboost.readthedocs.io](https://xgboost.readthedocs.io)
Lecture 11: Neural Networks (Part 1)

CIS 4190/5190
Fall 2022
Model Family for Neural Networks

• Modern view: Not a single model family

• Instead, a flexible framework for designing model families
Simple Example of Model Family

• **Feedforward neural network model family (for regression):**

\[
  f_{W,\beta}(x) = \beta^T g(Wx)
\]

• **Parameters:** Matrix \( W \in \mathbb{R}^{d \times k} \) and vector \( \beta \in \mathbb{R}^k \)
  
  • \( k \) is a hyperparameter called the **number of hidden neurons**

• Here, \( g: \mathbb{R} \rightarrow \mathbb{R} \) is a given **activation function**
  
  • It is applied componentwise in \( f_{W,\beta} \) (i.e., \( g \left( \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \right) = \begin{bmatrix} g(z_1) \\ g(z_2) \end{bmatrix} \))
  
  • **Example:** \( g(z) = \sigma(z) \) (where \( \sigma \) is the sigmoid function)
Simple Example of Model Family

- Feedforward neural network model family (for regression):

\[
 f_{W,\beta}(x) = \beta^T g(Wx)
\]
Simple Example of Model Family

• Feedforward neural network model family (for regression):

\[ f_{W,\beta}(x) = \beta^T g(Wx) \]

• What happens if \( g \) is linear? Recovers linear functions!

\[ f_{W,\beta}(x) = \beta^T g(Wx) = \beta^T Wx = \tilde{\beta}^T x \]

• **In general:** Linear regression over “features” \( g(Wx) \)
Modern View

• Not a single model family

• Instead, a **flexible framework** for **designing** model families
Modern View

• Feedforward neural network model family:

\[ f_{W,\beta}(x) = \beta^T g(Wx) \]
Modern View

- Feedforward neural network model family:

\[ f_{W, \beta}(x) = f_\beta \left( g \left( f_W(x) \right) \right) = f_\beta \circ g \circ f_W(x) \]
Modern View

• Each layer is a parametric function $f_{W_j}: \mathbb{R}^k \rightarrow \mathbb{R}^h$

• Compose sequentially to form model family:

$$f_W = f_{W_m} \circ \ldots \circ f_{W_1}$$

• Equivalently:

$$f_W(x) = f_{W_m} \left( \ldots \left( f_{W_1}(x) \right) \ldots \right)$$
Modern View

• Each **layer** is a parametric function $f_{W_j}: \mathbb{R}^k \rightarrow R^h$

• Can compose layers in other ways, e.g., concatenation:

$$f_W(x) = f_{W_1}(x) \oplus f_{W_2}(x)$$

• Here, we have defined

$$[z_1 \cdots z_d]^\top \oplus [z'_1 \cdots z'_{d'}]^\top = [z_1 \cdots z_d z'_1 \cdots z'_{d'}]^\top$$
Modern View

• Feedforward neural network model family (for regression):

\[ f_{W,\beta}(x) = f_\beta \circ g \circ f_W(x) \]
Modern View

• Feedforward neural network model family (for regression):

\[ f_{W,\beta}(x) = f_{\beta} \circ g \circ f_{W}(x) \]
Modern View

input layer

parameters (sometimes called “weights”)

hidden layer

nodes or “units” (i.e., components of a layer)

output layer

\[
\begin{align*}
\mathbf{x} & \xrightarrow{f_W} z^{(1)} \\
& \quad \xrightarrow{g} z^{(2)} \\
& \quad \xrightarrow{f_\beta} \hat{y}
\end{align*}
\]
Modern View

• Neural network with two hidden linear layers:

\[ f_{w_1,w_2,\beta}(x) = f_\beta \circ g \circ f_{w_2} \circ g \circ f_{w_1}(x) \]
Modern View

- Neural network with two hidden linear layers:

\[ f_{W_1,W_2,\beta}(x) = f_\beta \left( g \left( f_{W_2} \left( g \left( f_{W_1}(x) \right) \right) \right) \right) \]

Learn successively more “high-level” representations
What About Classification?

• **Recall:** For logistic regression, we choose the likelihood to be

\[ p_\beta(Y = 1 \mid x) = \frac{1}{1 + e^{-\beta^T x}} \]
What About Classification?

• **Recall**: For logistic regression, we choose the likelihood to be

\[ p_\beta(Y = 1 \mid x) = \sigma(\beta^T x) \]
What About Classification?

• For binary classification:

\[ p_{W,\beta}(Y = 1 \mid x) = \sigma(\beta^T g(Wx)) \]
What About Classification?

- For multi-class classification:

\[ p_{W,U}(Y = y \mid x) = \text{softmax}(Ug(Wx))_y \]
Neural Networks

• Pros
  • “Meta” strategy: Enables users to design model family
  • Design model families that capture symmetries/structure in the data (e.g., read a sentence forwards, translation invariance for images, etc.)
  • “Representation learning” (automatically learn features for certain domains)
  • More parameters!

• Cons
  • Very hard to train! (Non-convex loss functions)
  • Lots of parameters → need lots of data!
  • Lots of design decisions
Common Architectures

Feed-forward NNs

Recurrent NNs

Convolutional NNs

Transformer

Always coupled with word embeddings...