Machine Learning
V05: Ensemble Methods

Meta learning
Ensembles in practice
AdaBoost

Based on material from
Todd Holloway, Indiana University
Igor Labutov, Cornell University
Zhuowen Tu, University of California Los Angeles
Educational objectives

• **Know** when ensembles **should work** in practice

• Present **arguments how & why** ensembles **work** in practice

• **Know** and **apply** the **AdaBoost** algorithm to problems of classification and feature selection
1. META LEARNING
Ensembles are meta learning algorithms
Learning to combine learners

Ensembles in a nutshell
• Goal: **Combining** multiple **complementary classifiers** to increase performance
• Idea: Build different “experts”, and let them vote

Pros & cons
✓ Very **effective** in practice
✓ Good **theoretical guarantees**
✓ **Easy** to implement, not too much parameter tuning
✗ The result is not so transparent (**black box**)  
✗ **Not a compact** representation

Formal problem description
• Given $T$ binary classification hypotheses ($h_1,\ldots,h_T$), find a combined classifier with better performance of the form

$$
\hat{h}(x) = sgn\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)
$$

For regression, use average instead  
individual weight

Not to be confused with “learning to learn”, which also sometimes go by “meta learning”: [http://bair.berkeley.edu/blog/2017/07/18/learning-to-learn/ & appendix](http://bair.berkeley.edu/blog/2017/07/18/learning-to-learn/)
Why do they work? I
Intuitive thoughts

Intuition
• Utility of **combining diverse, independent opinions** in human decision-making
  E.g., stock portfolio diversity
• **Identifying** single best model (i.e., proper level of model complexity) is hard
  Example of Ockham’s 2\textsuperscript{nd} razor (“simplicity is always good”) being “blunt” → see [Domingos, 1998] and V03

Example of possible error reduction
• Suppose there are 25 **binary** base classifiers, each classifier has error rate $\varepsilon = 0.3$
• **Assume independence** among classifiers (i.e., classifiers are complementary)
• **Probability** that the final ensemble classifier makes a **wrong** prediction:

\[
p(\text{ensemble commits error}) = \sum_{r=13}^{25} \binom{25}{r} \cdot \varepsilon^r \cdot (1 - \varepsilon)^{25-r} \approx 0.06
\]

That is: combining 25 **completely independent classifiers with 70% accuracy** simply by majority vote yields a 94% accurate classifier! (→ see appendix for derivation)
Why do they work? II
Three fundamental reasons why they may work better

### Statistical
- **We cannot know** the best → so we average
- **Given** finite amount of data, many hypothesis typically appear equally good
- **Averaging** may be a better approximation to the true $f$

### Computational
- **We may not find** the best → so we average
- **Search** for $h$ is heuristic due to interesting $\mathcal{H}$’s being huge/infinite
- Strategy to avoid local minima: repeat with random restarts, construct an ensemble

### Representational
- **We cannot find** the best → so we average
- The desired target function may not be realizable using individual classifiers from $\mathcal{H}$
- It may be approximated by ensemble averaging
Why do they work? III  
In terms of bias and variance (→ see also V06)

Assume a regression task
- \( E_{MSE} = \text{bias}^2 + \text{variance} + \text{noise} \)

  - **Bias** problem:  
    E.g., \( \mathcal{H} \) used by particular learning method **doesn’t include** sufficient \( h \)’s (near true \( f \))
  - **Variance** problem:  
    E.g., \( \mathcal{H} \) is too “expressive” for the training data ⇒ selected \( h \) may **not generalize** well

Example: decision trees
- Small trees have high bias (i.e., too restricted \( \mathcal{H} \))
- Large trees have high variance  
  (i.e., very unstable decisions in the leaves)

Bias & variance in Ensembles
- **Bias remains equal** w.r.t. the base learners
- **Variance is reduced** with each added member

Attention: the bias-variance trade-off for classification has a very different (unintuitive) form ⇒ see appendix
Example: Bagging
Constructing for Diversity

Bootstrap Aggregating [Breiman, 1996]
• Almost always improves results if base learner is unstable (i.e., high variance)
• Why? \( \text{bias} \left( \hat{h}(x) \right) = \frac{1}{T} \sum_{t=1}^{T} \text{bias}(h_t(x)), \text{variance} \left( \hat{h}(x) \right) \approx \frac{1}{T} \text{variance}(h_t(x)) \)
  \( \Rightarrow \) usually, the more ensemble members, the better

Algorithm
1. for \( t := 1..T \)
2. \( X_t := \text{sample i.i.d. from } X \text{ with replacement} \)
3. \( h_t := \text{train any algorithm on } X_t \)
4. Return \( \hat{h} := \text{sgn} \left( \sum_{t=1}^{T} 1 \cdot h_t(x) \right) \)
  #(majority vote; for regression use average instead)
  \( \Rightarrow \) The process is remarkably simple (also to implement)
  \( \Rightarrow \) See appendix for Breiman’s extension into Random Forests®

Further Reading
2. ENSEMBLES IN PRACTICE
The Netflix Prize of 2006–2009
Ca. 3 years of challenging the global data science community

Supervised learning task
• Goal: Construct a classifier that, given a user and an unrated movie, correctly classifies that movie as either 1, 2, 3, 4, or 5 stars (i.e., predict rating by user)
• Input: Training data is set of users and ratings (1,2,3,4,5 stars) for movies
• Incentive: $1′000′000 for a 10% improvement over Netflix’s current movie recommender ($\text{EMSE}=0.9514$)

⇒ See http://www.netflixprize.com
Evolving results I
Low hanging fruits and slowed down progress

- **After 3 weeks**, at least **40 teams** had **improved** the Netflix classifier
- Top teams showed about 6% improvement
- However, **improvement slowed**:

Evolving results II
A leader board full of ensembles

Intermediate results
• Top team has posted a 8.5% improvement
• Ensemble methods are the best performers…
• …as we will see on the next slides
Details: Rookies

Quote

- “Thanks to Paul Harrison's collaboration, a simple mix of our solutions improved our result from 6.31 to 6.75”

<table>
<thead>
<tr>
<th>No Progress Prize candidates yet</th>
<th>--</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress Prize 2007 - RMSE &lt;= 0.8755</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>1: Korbell</td>
<td>0.8795</td>
<td>8.50</td>
</tr>
<tr>
<td>2: When Gravity and Dinosaurs Unite</td>
<td>0.877</td>
<td>8.38</td>
</tr>
<tr>
<td>3: Gravity</td>
<td>0.8743</td>
<td>8.10</td>
</tr>
<tr>
<td>4: Sasto</td>
<td>0.8746</td>
<td>8.07</td>
</tr>
<tr>
<td>5: Dinosaur Planet</td>
<td>0.8753</td>
<td>8.00</td>
</tr>
<tr>
<td>6: Mi qtl Toronto A</td>
<td>0.8737</td>
<td>7.84</td>
</tr>
<tr>
<td>7: Anja Palerak</td>
<td>0.8789</td>
<td>7.82</td>
</tr>
<tr>
<td>8: NIPS Reject</td>
<td>0.8808</td>
<td>7.42</td>
</tr>
<tr>
<td>9: Just a guy in a garage</td>
<td>0.8834</td>
<td>7.15</td>
</tr>
<tr>
<td>10: Ensemble Experts</td>
<td>0.8841</td>
<td>7.07</td>
</tr>
<tr>
<td>11: mathematical capital</td>
<td>0.8844</td>
<td>7.04</td>
</tr>
<tr>
<td>12: HowLowCanHeGo2</td>
<td>0.8847</td>
<td>7.01</td>
</tr>
<tr>
<td>13: The Thought Gang</td>
<td>0.8849</td>
<td>6.99</td>
</tr>
<tr>
<td>14: Reel Incubity</td>
<td>0.8855</td>
<td>6.93</td>
</tr>
<tr>
<td>15: strudelkama</td>
<td>0.8859</td>
<td>6.88</td>
</tr>
<tr>
<td>16: NIPS Submission</td>
<td>0.8861</td>
<td>6.86</td>
</tr>
<tr>
<td>17: Three Blind Mice</td>
<td>0.8869</td>
<td>6.78</td>
</tr>
<tr>
<td>18: TrainOnTest</td>
<td>0.8859</td>
<td>6.73</td>
</tr>
<tr>
<td>19: Brett Dean</td>
<td>0.8895</td>
<td>6.75</td>
</tr>
<tr>
<td>20: Paul Harrison</td>
<td>0.8872</td>
<td>6.75</td>
</tr>
<tr>
<td>21: ATTEAM</td>
<td>0.8873</td>
<td>6.74</td>
</tr>
<tr>
<td>22: wyozconsulting.com</td>
<td>0.8874</td>
<td>6.73</td>
</tr>
<tr>
<td>23: ICML submission</td>
<td>0.8875</td>
<td>6.72</td>
</tr>
<tr>
<td>24: Efretko</td>
<td>0.8877</td>
<td>6.70</td>
</tr>
<tr>
<td>25: Kitty</td>
<td>0.8881</td>
<td>6.65</td>
</tr>
<tr>
<td>26: Secondary Results</td>
<td>0.8884</td>
<td>6.62</td>
</tr>
<tr>
<td>27: Birgit Kraft</td>
<td>0.8885</td>
<td>6.61</td>
</tr>
</tbody>
</table>
Details: Arek Paterek

Quote

- “My approach is to combine the results of many methods (also two-way interactions between them) using linear regression on the test set. The best method in my ensemble is regularized SVD with biases, post processed with kernel ridge regression”

Details: University of Toronto

Quote

• “When the predictions of multiple RBM models and multiple SVD models are linearly combined, we achieve an error rate that is well over 6% better than the score of Netflix’s own system.”

Details: Gravity

Quote

- Table 5: Best results of single approaches and their combinations

<table>
<thead>
<tr>
<th>Method/Combination</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF</td>
<td>0.9190</td>
</tr>
<tr>
<td>NB</td>
<td>0.9313</td>
</tr>
<tr>
<td>CL</td>
<td>0.9606</td>
</tr>
<tr>
<td>NB + CL</td>
<td>0.9275</td>
</tr>
<tr>
<td>MF + CL</td>
<td>0.9137</td>
</tr>
<tr>
<td>MF + NB</td>
<td>0.9089</td>
</tr>
<tr>
<td>MF + NB + CL</td>
<td>0.9089</td>
</tr>
</tbody>
</table>

[home.mit.bme.hu/~gtakacs/download/gravity.pdf]
Details: When Gravity and Dinosaurs Unite

Quote

• “Our common team **blends the result** of team Gravity and team Dinosaur Planet.”
Details: BellKor / KorBell

Quote

• “Our final solution (RMSE=0.8712) consists of blending 107 individual results.”
Evolving results III
Final results

The winner was an **ensemble of ensembles** (including BellKor)
- Gradient boosted decision trees [http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf]

→ **Hint:** Ensembles still win competitions, but **Deep Learning** has **better** performance for unstructured data (→ see later and [https://www.import.io/post/how-to-win-a-kaggle-competition/](https://www.import.io/post/how-to-win-a-kaggle-competition/))

→ The **winner model was never used** in Netflix’ practice due to its complexity

E.g. on *Kaggle, pattern recognition benchmarks like ImageNet, etc.*
**XGBoost: A scalable tree boosting system**  
[Chen & Guestrin, 2016] → using gradient boosting, see appendix

A skillfully engineered, highly optimized implementation
- Used by 17/29 winning teams on Kaggle 2015
- Open source (Python, R, Spark, …): [https://github.com/dmlc/xgboost](https://github.com/dmlc/xgboost)
- Scalable: 10× faster than usual implementations, scales to \( \sim 10^9 \) training points
  - Massive use of parallelization/distribution (e.g. on Hadoop/Spark, but also on desktop)

Algorithmic novelties
- Distributed approximate best split finding (“weighted quantile sketch” using quantile statistics)
- **Exploit sparsity** (induced by missing values/one-hot encoding → via default directions for branching)

Parallelization Cache-aware access (for gradient statistics)
- Efficient out-of-core computation (i.e., computation on data not fitting into main memory)

General tricks for tree boosting
- Use aggressive sub-sampling (e.g., selecting only 50% of the data)
- Using column sub-sampling prevents over-fitting even more so than row sub-sampling
3. ADABOOST
Boosting

General idea
- **Boost** the performance of weak learners (error slightly > chance) **iteratively**
- **Make** currently misclassified examples more important, then combine hypotheses
  - Each stage (additively) corrects shortcomings of previous stage by reweighting, then **majority vote**
- Origins in computer science: [Kearns & Valiant, 1988] (as opposed to Bagging: statistics)

**Adaptive Boosting** algorithm [Freund & Schapire, 1997]
- Weak learner: decision stump (= decision tree of height 1; but generalizable to others)
  - Important: weak learners have skill but remain weak (to not lose the ensemble effect)

initialize weights: $w_i := \frac{1}{N}$ # each sample gets same weight

for $t := 1..T$

$h_t := \text{train decision stump on the } x_i, \text{ weighted by the } w_i$

$\varepsilon_t := \frac{\sum_{i=1}^{N} w_i \cdot I(y_i \neq h_t(x_i))}{\sum_{i=1}^{N} w_i}$ # compute error; $I()$ is the identity function

$\alpha_t := \log \left( \frac{1-\varepsilon_t}{\varepsilon_t} \right)$ # compute influence of weak learner

$w_i := w_i \cdot e^{\alpha_t \cdot I(y_i \neq h_t(x_i))}$ # increase weight by exp(influence) in case of error

return $\hat{h} := sgn(\sum_{t=1}^{T} \alpha_t \cdot h_t(x))$ # majority vote
Example run

Source
- [Elder, 2007]: «From Trees to Forests and Rule Sets – A Unified Overview of Ensemble Methods»
AdaBoost in practice

Pros & cons
✓ Very little code
✓ Reduces bias & variance
✓ Still learns when others overfit → margin optimization
✗ Sensitive to noise and outliers

Implementation choices
• A good start for implementation is the variant “AdaBoost.M1” from [Frank & Witten, 2005], combined with ideas from “Real AdaBoost.MH” of [Schapire & Singer, 1999]
• For cost-sensitive binary classification, use “AdaC2” from [Sun et al., 2007]

Further reading
• [Freund & Schapire, 1997]: «A decision-theoretic generalization of on-line learning and an application to boosting»
• [Sun et al., 2007]: «Cost-Sensitive Boosting for Classification of Imbalanced Data»
• [Frank & Witten, 2005]: «Data Mining - Practical Machine Learning Tools and Techniques», 2nd Ed.
• [Schapire & Singer, 1999]: «Improved Boosting Algorithms Using Confidence-rated Predictions»
Example application: Real-time face detection
AdaBoost as a feature selector

Viola & Jones face detector
- The first method for object detection in images with human-like performance (today outperformed by deep learning approaches)
- AdaBoost applied to >160’000 features
- First $k$ selected features of decision stumps are deemed meaningful
- Trained on very unbalanced data (faces ↔ non-faces)

Further reading → see appendix
- [Viola & Jones, 2001]: “Rapid object detection using a boosted cascade of simple features”
- [Viola & Jones, 2003]: “Robust Real-Time Face Detection”

Review

• Ensembles can be seen as **meta learners** (operating on learners, not data): **learning** to make the **best of many base learners**

• Building ensembles can be as easy as **Bagging**: train any $T$ **classifiers** on different **bootstrap** samples, then take a (classification:) **majority vote** or (regression:) **average**

• Ensembles work because they use **averaging in a clever way**: **reduce variance**, reach $\hat{h} \notin \mathcal{H}$, **overcome small data** sets

• Ensembles have been **very successful** in the past; it is good advice to **always build an ensemble of complementary models** as the final classifier

• **AdaBoost** is very **immune to overfitting** and can be used for feature selection (→ see appendix)
P04.3: Building ensembles

Work through exercise P04.3

• Goal is to build a final classifier for SPAM classification
• Which one of different algorithms performs best?
• Is a combination beneficial on this task?
APPENDIX

- More on ensembles and error analysis
- Random Forest® and gradient boosting
- The Viola-Jones face detector
Learning to learn

The Auto-sklearn pipeline approach

- 2 times winner of AutoML challenge (2015/16 & 2017/18)
- Utilizes good initialization by starting from a well performing model on a similar dataset seen as seen during meta learning
- Uses Bayesian optimization of pipeline and hyperparameters to tweak this model
- Finally builds an ensemble of best candidates


See also: Tuggener et al., “Automated Machine Learning in Practice: State of the Art and Recent Results”, Proc. 6th Swiss Conference on Data Science (SDS), 2019
Derivation: Ensemble error of $t$ independent binary classifiers

- Suppose there are $t$ independent base classifiers, each classifier has error rate $\epsilon$

- They form an ensemble via majority voting: \[ \left\lfloor \frac{t}{2} \right\rfloor \] base classifiers have to be correct for the ensemble to be correct

- Let $E_r$ be the event that $r$ out of $t$ base classifiers vote incorrectly:
  Its probability follows a binomial distribution $p(E_r) = \binom{t}{r} \cdot \epsilon^r \cdot (1 - \epsilon)^{t-r}$

- Let $E$ be the event that the whole ensemble is wrong (i.e., at least $\left\lfloor \frac{t}{2} \right\rfloor$ incorrect votes):
  Its probability is given by $p(E) = \sum_{r=\left\lfloor \frac{t}{2} \right\rfloor}^{t} p(E_r)$

Reasoning
- $E$ occurs if $\left\lfloor \frac{t}{2} \right\rfloor$ base classifiers are wrong, or if $\left\lfloor \frac{t}{2} \right\rfloor + 1$ base classifiers are wrong, or if ... $t$ base classifiers are wrong
- Assuming independence among these events, their probabilities are added
Discussion: Bias-variance trade-off for 0/1 loss
Going from regression to classification

Definitions: Bias and variance of a learner w.r.t a single instance $x$ [Domingos, 2000]

- $bias :=$ deviation of best possible prediction from main prediction
- $variance :=$ average deviation (over all training sets) from actual to main prediction

Regression

- The **bias-variance trade-off** has originally been defined for regression problems
- Typical loss function is the **mean squared error** (MSE)
  - $L_{MSE} = bias^2 + variance + noise$ (→ see V03)

Classification

- Usually binary classification is studied in depth first → result may then be extended to multi-class
- Binary classification uses **classification error** as its typical loss function (a.k.a. 0/1 loss)
  - The main prediction is the most frequent prediction; we subsequently ignore the additive noise term
  - $L_{0/1} = bias + variance$ in case of $bias = 0$ (i.e., classifier is correct $> 50\%$ of the time)
  - $L_{0/1} = bias - variance$ in case of $bias = 1$ (i.e., classifier’s accuracy is $\leq 50\%$)
Discussion: Bias-variance trade-off for 0/1 loss
Counter-intuitive implications

Consequences for classification

- Bias and variance have a **complicated, multiplicative interaction** [Friedman, 1997] (→ not directly visible in the form shown on the last slide due to the 2 cases)
- Good classifiers become better with less variance; **bad classifiers become better with more variance!**
- This explains why **highly unstable classifiers** (e.g., decision trees; kNN in high dimensions; naïve Bayes) **work well in practice**
- Casting classification as a regression problem by **estimating class probabilities** instead often **doesn’t pay off**:
  - Good regression results don’t imply good classification performance
  - Reason: Different behavior of errors

Further reading

- [Domingos, 2000]: «A Unified Bias-Variance Decomposition for Zero-One and Squared Loss»
- [Friedman, 1997]: «On Bias, Variance, 0/1-Loss, and the Curse of Dimensionality»
Random Forest®
A brief description

Build a majority-voting ensemble of decision trees; for each tree,
- Choose a stratified training set of \( n \) out of \( N \) instances by sampling with replacement
- At every level,
  - choose a random feature set (with replacement) of \( m \) out the \( p \) attributes
  - choose the best split among those attributes
- **No pruning** of the branches takes place

Advantages
- Fast training, parallelizable application
- High independence of base classifiers \( \rightarrow \) nearly no overfitting
- Few hyper parameters
- Applicable to large quantities of \( N, p \) and #classes
  \( \rightarrow \) Very good out-of-the-box method

Further reading
- [Breiman 2001]: «Random Forests». Machine Learning 45 (1), 5-32
From AdaBoost to gradient boosting

Recall: In AdaBoost, "shortcomings" are identified by high-weight data points.

A brief history of modern boosting (selective, shortened)

1. **Invention**: AdaBoost, the first successful boosting algorithm
   [Freund et al., 1996], [Freund & Schapire, 1997]
2. **Translation**: Formulation as gradient descent with special loss function (→ cp. V02)
   [Breiman et al., 1998], [Breiman, 1999]
3. **Generalization**: Gradient boosting in order to handle a variety of loss functions
   [Friedman et al., 2000], [Friedman, 2001]

→ For a great example of cross-disciplinary fertilization, see
   Breiman, “Arcing classifiers (with discussion and a rejoinder by the author)”, 1998

In gradient boosting, "shortcomings" are identified by gradients
- Gradients of what? Why? → see next slides
Intuition for gradient boosting

Setup
• For ease of discussion we change the setting from (binary) classification to regression (i.e., real-valued labels)
• Results are again applicable to classification (but not intuitively as straight-forward)

Let’s play a game
• You are given data \{(x_1, y_1), \ldots, (x_N, y_N)\} and the task to fit model \(\hat{h}(x)\)

  \[ \Rightarrow \text{minimize squared loss } \ell(y, h(x)) = \frac{1}{2} (y - h(x))^2 \]
• Suppose a friend helps by giving you an initial model \(F(x)\) (a regression tree)

  \[ \Rightarrow \text{You check his model and find the model is good but not perfect (e.g. } F(x_1) = 0.8 \text{ while } y_1 = 0.9) \]
• Rule: \(F(x)\) must not be changed in any way, but another model might be added

  \[ \Rightarrow \text{i.e. } \hat{h}(x) = F(x) + h(x) \]
• How to train \(h(x)\)?

  We want this to be true

  \[
  \begin{align*}
  F_1(x_1) + h(x_1) &= y_1 \\
  F(x_N) + h(x_N) &= y_N
  \end{align*}
  \]

  \[ \Rightarrow h(x_1) = y_1 - F(x_1) \]

  \[ \Rightarrow h(x_N) = y_N - F(x_N) \]

  Equivalently, we can fit the new regression tree \(h\) to:
Intuition for gradient boosting (contd.)

Simple ensemble solution
- The \( y_i - F(x_i) \) are called residuals
  - These are the parts that the initial model \( F \) cannot do well
  - The role of \( h \) is to compensate the shortcomings of \( F \)
- If the new model \( F + h \) is still not satisfactory, we can add another regression tree...

How is this related to gradient descent?
- Gradient Descent: **Minimize** function \( J \) by **moving** into **opposite direction** of the gradient
  \[
  \theta_i^{\text{new}} = \theta_i^{\text{old}} - \alpha \frac{\partial J}{\partial \theta_i^{\text{old}}}
  \]
  i.e., \( J = L \)

- Want to **minimize loss** function: \( L = \sum_{i=1}^{N} \ell(y_i, F(x_i)) = \sum_{i=1}^{N} \frac{1}{2} (y_i - F(x_i))^2 \)
  - \( F(x_i) \) are the parameters of \( L \), so we can take derivatives:
    \[
    \frac{\partial L}{\partial F(x_i)} = \frac{\partial}{\partial F(x_i)} \sum_{i=1}^{N} \ell(y_i, F(x_i)) = \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i
    \]
- That is: We can interpret residuals as negative gradients
  \[
  y_i - F(x_i) = -\frac{\partial L}{\partial F(x_i)}
  \]
Gradient boosting of regression trees

Algorithm

- Gradient boosting for regression

Start with an initial model, e.g. \( F = \frac{\sum_{i=1}^{N} y_i}{N} \) (always predict mean value)

repeat until convergence

\[ -g(x_i) = -\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \]

fit regression tree \( h \) to \(-g(x_i)\)

\( F := F + \alpha h \)  \# \( \alpha \) is a tunable learning rate, e.g. = 1

True for \( \ell = \text{squared loss} \)
- Residual \( \Leftrightarrow \) negative gradient
- Fit \( h_i \) to residual \( \Leftrightarrow \) fit \( h_i \) to negative gradient
- Update \( h_i \) based on residual \( \Leftrightarrow \) update \( h_i \) based on negative gradient

\( \Rightarrow \) So we are actually updating our model using gradient descent!

Advantage of gradient descent formulation

- Allows considering other loss functions (e.g. more outlier-robust, domain-specific, …)

\( \Rightarrow \) Derive the corresponding algorithms in the same way
Extension to (multiclass) classification

Model
• Each class $c$ has its own model $F_c(x)$ (binary classification tree, emitting 0/1)
• Use outputs to compute class probabilities: $P_c(x) = \frac{e^{F_c(x)}}{\sum_i e^{F_i(x)}}$ (softmax)
  ➔ Final classification = class with highest probability

Loss function per data point
• Turn the label $y_i$ into a (true) probability distribution $Y_c(x_i)$
• Calculate predicted probability distribution $P_c(x_i)$
  ➔ Based on current models $F_c(x_i)$
• Calculate difference between true and predicted probability distribution
  ➔ Use e.g. KL-divergence as loss

Overall objective
• Do gradient descent to make true and predicted distribution as close as possible $\forall x_i$
• We achieve this goal by adjusting our models $F_c$
AdaBoost for face detection
A detailed example of a boosted decision stumps application

Challenges
- **Slide a window** across image and evaluate a face model at every location & scale
  - Sliding window detector must evaluate tens of thousands of location/scale combinations
- Faces are **rare**: 0–10 per image
  - For computational efficiency, we should try spending as little time as possible on non-face windows
    - A megapixel image has $\sim 10^6$ pixels and a comparable number of candidate face locations
  - To avoid having a false positive in every image, the false positive rate has to be less than $10^{-6}$

The Viola-Jones face detector [Viola & Jones, 2001]
- A seminal approach to real-time object detection
  - Training is slow, but detection is very fast
- Key ideas
  - Integral images for fast feature evaluation
  - Boosting for feature selection amongst $\sim 10^5$ candidates
  - Attentional cascade for fast & accurate rejection of non-face windows
Rectangular facial features
...and their efficient calculation via the integral image

Pixel-based features for face detection
- Reminiscent of Haar wavelets
- Simple **sum of pixel intensities** within rectangular regions resemble typical shading patterns of faces

Integral images (ii)
- Let each pixel be the **sum** of all pixels left and above

Computing sums of pixels within a rectangle using ii
- \( \text{sum} = ii_A - ii_B - ii_C + ii_D \)
- Needs **only 3 additions** for any size of rectangle (constant time)
Feature selection via AdaBoost
Slide adapted from Grauman & Leibe’s AAAI’08 tutorial

Size of feature space
• Ca. 160,000 distinct rectangular features per detection window (via scaling/translation)
  ➔ Which ones are good? What is a good subset?

Finding a good succession of features
• Start: Select the single rectangle feature & threshold that best separates faces/non-faces

Resulting weak classifier:
\[ h_t(x) = \begin{cases} 
+1 & \text{if } f_t(x) > \theta_t \\
-1 & \text{otherwise} 
\end{cases} \]

➔ Continue using AdaBoost

Outputs of a possible rectangle feature on faces and non-faces.
Training the boosting classifier
Incorporating feature selection

Training set contains face and non-face examples
- **5000 faces** (frontal, many variations among illumination/pose, rescaled to 24 × 24)
- **300 million non-faces** (extracted from 9'500 non-face images)
- Faces are normalized (scale, translation)
- Initially, all have equal weights

For each round of boosting:
- Evaluate each rectangle filter on each example, select best threshold
- Select best filter/threshold combination
- Reweight examples
→ Computational complexity: $O(\text{rounds} \times \text{examples} \times \text{features})$

Result
- A 200-feature classifier can yield **95% detection rate** and a false positive rate of 1 in 14084
→ Not yet **good** enough for practice!

False alarm rate of $O(10^{-5})$, but $O(10^{-6})$ needed
Removing false alarms while retaining high detection rate

Attentional Cascade
- **Start** with a **simple** classifier (2 features)
  - Rejecting many of the **negative** sub-windows while detecting almost all positive sub-windows
- **Positive** response from the first classifier triggers the evaluation the **next** classifier, etc.
  - Subsequent classifiers get more complex, hence longer runtime but lower false alarm rate
- A negative outcome at any point leads to the immediate rejection of the sub-window
- **Training:**
  - Keep **adding** features to current stage until its target rates (TP, FP) have been **met**
  - If overall FP is **not low** enough, then add another stage
  - Use false positives from current stage as the negative training examples for the next stage

Detection rate (TP) vs. false alarm rate (FP) for chained classifiers
- Found by multiplying the respective rates of the individual stages
  - TP of 0.9 and FP of $\sim 10^{-6}$ can be achieved with a **10-stage** cascade: each stage having
    - TP of 0.99 ($0.99^{10} \approx 0.9$)
    - FP of $\sim 0.3$ ($0.3^{10} \approx 6 \times 10^{-6}$)
Final result of Viola-Jones face detection

After some more engineering…

• Variance normalization of pixel intensities to cope with different lighting
• Merging multiple detections
• Multi-scale detection by scaling the detector (factor of 1.25 yields good resolution)

Lasting effect

• Got applied to more visual detection problems
  ➔ facial feature localization, profile faces, male/female image classification, audio fingerprinting, …
• Solved the problem of face detection in real time (e.g. for digicams)
  ➔ available in OpenCV (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html)
• One of the first mind-blowing computer vision applications before deep learning trend