Data Mining II
Ensembles

Heiko Paulheim
Introduction

• “Wisdom of the crowds”
  – a single individual cannot know everything
  – but together, a group of individuals knows a lot

• Examples
  – Wikipedia
  – Crowdsourcing
  – Prediction

http://xkcd.com/903/
Introduction

• “SPIEGEL Wahlwette” (election bet) 2013
  – readers of SPIEGEL Online were asked to guess the federal election results
  – average across all participants:
    • only a few percentage points error for final result
    • conservative-liberal coalition cannot continue

https://lh6.googleusercontent.com/-U9DXTTcT-PM/UgsdSzdV3JI/AAAAAAAAFKs/GsRydeldasg/w800-h800/Bildschirmfoto+2013-08-14+um+07.56.01.png
Introduction

• “Who wants to be a Millionaire?”
• Analysis by Franzen and Pointner (2009):
  – “ask the audience” gives a correct majority result in 89% of all cases
  – “telephone expert”: only 54%

Ensembles

• So far, we have addressed a learning problem like this:

• Ensembles:
  – wisdom of the crowds for learning operators
  – instead of asking a single learner, combine the predictions of different learners
Ensembles

• Prerequisites for ensembles: accuracy and diversity
  – different learning operators can address a problem (accuracy)
  – different learning operators make different mistakes (diversity)

• That means:
  – predictions on a new example may differ
  – if one learner is wrong, others may be right

• Ensemble learning:
  – use various base learners
  – combine their results in a single prediction
Voting

• The most straightforward approach
  – classification: use most-predicted label
  – regression: use average of predictions

• We have already seen this
  – k-nearest neighbors
  – each neighbor can be regarded as an individual classifier
Voting in RapidMiner

- Vote operator uses different base learners
Voting in RapidMiner

- Accuracy in this example:
  - Naive Bayes: 0.71
  - Ripper: 0.71
  - k-NN: 0.81
- Voting: 0.91
Why does Voting Work?

• Suppose there are 25 base classifiers
  – Each classifier has an accuracy of 0.65, i.e., error rate $\varepsilon = 0.35$
  – Assume classifiers are independent
    • i.e., probability that a classifier makes a mistake does not depend on whether other classifiers made a mistake
    • **Note:** in practice they are not independent!

• Probability that the ensemble classifier makes a wrong prediction
  – The ensemble makes a wrong prediction if the majority of the classifiers makes a wrong prediction
  – The probability that 13 or more classifiers are wrong is

\[
\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} \approx 0.06 \ll \varepsilon
\]
Why does Voting Work?

• In theory, we can lower the error infinitely just by adding more base learners
  - The formula only holds for independent base learners
    - It is hard to find many truly independent base learners
      - ...at a decent level of accuracy
  - Recap: we need both accuracy and diversity

\[ \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1 - \varepsilon)^{25-i} \approx 0.06 \ll \varepsilon \]
Recap: Overfitting and Noise

Likely to overfit the data
Bagging

• Biases in data samples may mislead classifiers
  – overfitting problem
  – model is overfit to single noise points

• If we had different samples
  – e.g., data sets collected at different times, in different places, …
  – …and trained a single model on each of those data sets…
  – only one model would overfit to each noise point
  – voting could help address these issues

• But usually, we only have one dataset!
Bagging

• Models may differ when learned on different data samples

• Idea of bagging:
  – create samples by picking examples \textit{with replacement}
  – learn a model on each sample
  – combine models

• Usually, the same base learner is used

• Samples
  – differ in the subset of examples
  – replacement randomly re-weights instances (see later)
Bagging: illustration

Training Data

Data1 → Learner1 → Model1

Data2 → Learner2 → Model2

... → Learner m → Model m

Model Combiner

Final Model
Bagging: Generating Samples

- Generate new training sets using sampling with replacement (bootstrap samples)

<table>
<thead>
<tr>
<th>Original Data</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagging (Round 1)</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>Bagging (Round 2)</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Bagging (Round 3)</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

- some examples may appear in more than one set
- some examples will appear more than once in a set
- for each set of size $n$, the probability that a given example appears in it is
  \[ \Pr(x \in D_i) = 1 - \left(1 - \frac{1}{n}\right)^n \rightarrow 0.6322 \]
  - i.e., on average, less than 2/3 of the examples appear in any single bootstrap sample
Bagging in RapidMiner

- Bagging operator uses a base learner
- Number and ratio of samples can be specified
Bagging in RapidMiner

- Accuracy in this example:
  - Ripper alone: 0.71
  - Ripper with bagging (10x0.5): 0.86
Bagging in RapidMiner

• 10 different rule models are learned:

```
if attribute_12 \leq 0.168 \text{ and } attribute_25 > 0.553 \text{ then Rock (28 / 0)}
if attribute_28 > 0.854 \text{ then Mine (2 / 23)}
if attribute_31 \leq 0.443 \text{ and } attribute_30 > 0.246 \text{ then Mine (0 / 16)}
if attribute_4 \leq 0.061 \text{ then Rock (16 / 0)}
else Mine (2 / 5)
```
correct: 88 out of 92 training examples.
Variant of Bagging: Randomization

• Randomize the learning algorithm instead of the input data

• Some algorithms already have a random component
  – e.g. initial weights in neural net

• Most algorithms can be randomized, e.g., greedy algorithms:
  – Pick from the $N$ best options at random instead of always picking the best options
  – e.g.: test selection in decision trees or rule learning

• Can be combined with bagging
Random Forests

- A variation of bagging with decision trees
- Train a number of individual decision trees
  - each on a random subset of examples
  - only analyze a random subset of attributes for each split
    *(Recap: classic DT learners analyze all attributes at each split)*
  - usually, the individual trees are left unpruned
Paradigm Shift: Many Simple Learners

• So far, we have looked at learners that are as good as possible

• Bagging allows a different approach
  – several simple models instead of a single complex one
  – Analogy: the SPIEGEL poll (mostly no political scientists, nevertheless: accurate results)
  – extreme case: using only decision stumps

• Decision stumps:
  – decision trees with only one node
Bagging with Weighted Voting

• Some learners provide confidence values
  – e.g., decision tree learners
  – e.g., Naive Bayes

• Weighted voting
  – use those confidence values for weighting the votes
  – some models may be rather sure about an example, while others may be indifferent
Weighted Voting with Decision Stumps

- Weights: confidence values in each leaf

lower confidence that it is mine (weight = 0.6)

high confidence that it is rock (weight = 1.0)
Intermediate Recap

• What we've seen so far
  – ensembles often perform better than single base learners
  – simple approach: voting, bagging

• More complex approaches coming up
  – Boosting
  – Stacking

• Boosting requires learning with *weighted instances*
  – we'll have a closer look at that problem first
Intermezzo: Learning with Weighted Instances

• So far, we have looked at learning problems where each example is equally important

• Weighted instances
  – assign each instance a weight (*think*: importance)
  – getting a high-weighted instance wrong is more expensive
  – accuracy etc. can be adapted

• Example:
  – data collected from different sources (e.g., sensors)
  – sources are not equally reliable
    • we want to assign more weight to the data from reliable sources
Intermezzo: Learning with Weighted Instances

• Two possible strategies of dealing with weighted instances

• Changing the learning algorithm
  – e.g., decision trees, rule learners: adapt splitting/rule growing heuristics, example on following slides

• Duplicating instances
  – an instance with weight n is copied n times
  – simple method that can be used on all learning algorithms
Recap: Accuracy

- Most frequently used metrics:

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]

Error Rate = 1 – Accuracy

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=Yes</td>
<td>Class=Yes</td>
</tr>
<tr>
<td>Class=Yes</td>
<td>TP</td>
</tr>
<tr>
<td>Class=No</td>
<td>FP</td>
</tr>
</tbody>
</table>
Accuracy with Weights

• Definition of accuracy

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]

• Without weights, TP, FP etc. are *counts* of instances

• With weights, they are *sums of their weights*
  – classic TP, FP etc. are the special case where all weights are 1
Adapting Algorithms: Decision Trees

• Recap: Gini index as splitting criterion

\[ GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2 \]

• The probabilities are obtained by counting examples
  – Again, we can sum up weights instead

• The same works for rule-based classifiers and their heuristics
Adapting Algorithms: k-NN

• Standard approach
  – use average of neighbor predictions

• With weighted instances
  – weighted average
Back to Ensembles: Boosting

• Idea of boosting
  – train a set of classifiers, one after another
  – later classifiers focus on examples that were misclassified by earlier classifiers
  – weight the predictions of the classifiers with their error

• Realization
  – perform multiple iterations
    • each time using different example weights
  – weight update between iterations
    • *increase* the weight of *incorrectly* classified examples
    • so they become more important in the next iterations (misclassification errors for these examples count more heavily)
  – combine results of all iterations
    • weighted by their respective error measures
Boosting – Algorithm AdaBoost.M1

1. initialize example weights $w_i = 1/N$ ($i = 1..N$)

2. for $m = 1$ to $t$ // $t$ ... number of iterations
   a) learn a classifier $C_m$ using the current example weights
   b) compute a weighted error estimate
      \[ \text{err}_m = \frac{\sum w_i \text{of all incorrectly classified } e_i}{\sum_{i=1}^{N} w_i} \]
      \[ = 1 \quad \text{because weights are normalized} \]
   c) if $\text{err}_m > 0.5$ → exit loop
   d) compute a classifier weight
      \[ \alpha_m = \frac{1}{2} \ln \left( \frac{1 - \text{err}_m}{\text{err}_m} \right) \]
   e) for all correctly classified examples $e_i$: $w_i \leftarrow w_i e^{-\alpha_m}$
   f) for all incorrectly classified examples $e_i$: $w_i \leftarrow w_i e^{\alpha_m}$
   g) normalize the weights $w_i$ so that they sum to 1

3. for each test example
   a) try all classifiers $C_m$
   b) predict the class that receives the highest sum of weights $\alpha_m$
Illustration of the Weights

- Classifier Weights $\alpha_m$
  - differences near 0 or 1 are emphasized
- Good classifier
  $\rightarrow$ highly positive weight
- Bad classifier
  $\rightarrow$ highly negative weight
- Classifier with error 0.5
  $\rightarrow$ weight 0
  $\rightarrow$ this is equal to guessing!
Illustration of the Weights

- Example Weights
  - multiplier for correct and incorrect examples
  - depending on error

- Later iterations need to focus on examples that are
  - Incorrectly classified by a good classifier
  - Correctly classified by a bad classifier
Boosting – Error Rate Example

- boosting of decision stumps on simulated data

from Hastie, Tibshirani, Friedman: The Elements of Statistical Learning, Springer Verlag 2001
Toy Example

\[ D_1 \]

(taken from Verma & Thrun, Slides to CALD Course CMU 15-781, Machine Learning, Fall 2000)
Round 1

\[\varepsilon_1 = 0.30\]
\[\alpha_1 = 0.42\]
Round 2

\[ \varepsilon_2 = 0.21 \]
\[ \alpha_2 = 0.65 \]
Round 3

$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$
Final Hypothesis

\[ H_{\text{final}} = \text{sign} \left( \begin{array}{c} 0.42 \\ + 0.65 \\ + 0.92 \end{array} \right) \]
Hypothesis Space of Ensembles

- Each learner has a *hypothesis space*
  - e.g., decision stumps: a linear separation of the dataset

- The hypothesis space of an ensemble
  - can be larger than that of its base learners

- Example: bagging with decision stumps
  - different stumps → different linear separations
  - resulting hypothesis space also allows polygon separations
Boosting in RapidMiner

• Just like voting and bagging
Experimental Results on Ensembles

- Ensembles have been used to improve generalization accuracy on a wide variety of problems.
- On average, Boosting provides a larger increase in accuracy than Bagging.
  - Boosting on rare occasions can degrade accuracy.
  - Bagging more consistently provides a modest improvement.
- Boosting is particularly subject to over-fitting when there is significant noise in the training data.
  - Subsequent learners over-focus on noise points.

(Freund & Schapire, 1996; Quinlan, 1996)
Back to Combining Predictions

• Voting
  – each ensemble member votes for one of the classes
  – predict the class with the highest number of vote (e.g., bagging)

• Weighted Voting
  – make a weighted sum of the votes of the ensemble members
  – weights typically depend
    • on the classifier's confidence in its prediction
      (e.g., the estimated probability of the predicted class)
    • on error estimates of the classifier (e.g., boosting)

• Stacking
  – Why not use a classifier for making the final decision?
  – training material are the class labels of the training data and the
    (cross-validated) predictions of the ensemble members
Stacking

• Basic Idea:
  – learn a function that combines the predictions of the individual classifiers

• Algorithm:
  – train \( n \) different classifiers \( C_1 \ldots C_n \) (the base classifiers)
  – obtain predictions of the classifiers for the training examples
  – form a new data set (the meta data)
    • classes
      – the same as the original dataset
    • attributes
      – one attribute for each base classifier
      – value is the prediction of this classifier on the example
  – train a separate classifier \( M \) (the meta classifier)
Stacking (2)

- Example:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11}$</td>
<td>$t$</td>
</tr>
<tr>
<td>$x_{21}$</td>
<td>$f$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{n_{ea}}$</td>
<td>$t$</td>
</tr>
</tbody>
</table>

training set

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>...</th>
<th>$C_{n_c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>...</td>
<td>$f$</td>
</tr>
<tr>
<td>$f$</td>
<td>$t$</td>
<td>...</td>
<td>$t$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$t$</td>
<td>$f$</td>
<td>...</td>
<td>$t$</td>
</tr>
</tbody>
</table>

predictions of the classifiers

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>...</th>
<th>$C_{n_c}$</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>$t$</td>
<td>...</td>
<td>$f$</td>
<td>$t$</td>
</tr>
<tr>
<td>$f$</td>
<td>$t$</td>
<td>...</td>
<td>$t$</td>
<td>$f$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$f$</td>
<td>$f$</td>
<td>...</td>
<td>$t$</td>
<td>$t$</td>
</tr>
</tbody>
</table>

training set for stacking

- Using a stacked classifier:
  - try each of the classifiers $C_1...C_n$
  - form a feature vector consisting of their predictions
  - submit these feature vectors to the meta classifier $M$
Stacking and Overfitting

• Consider a dumb base learner D, which works as follows:
  – during training: store each training example
  – during classification: if example is stored, return its class
    otherwise: return a random prediction

• If D is used along with a number of classifiers in stacking, what will the meta classifier look like?
  – D is perfect on the training set
  – so the meta classifier will say: always use D's result

Implementation in RapidMiner :-(

do you know that classifier?
Stacking and Overfitting

• Solution 1: split dataset (e.g., 50/50)
  – use one portion for training the base classifiers
  – use other portion to train meta model

• Solution 2: cross-validate base classifiers
  – train classifier on 90% of training data
  – create features for the remaining 10% on that classifier
  – repeat 10 times

• The second solution is better in most cases
  – uses whole dataset for meta learner
  – uses 90% of the dataset for base learners

X-Stacking in Mannheim RapidMiner Toolbox :-(
Stacking in RapidMiner

- Looks familiar again
  - we need a set of base learners (like for voting)
  - and a learner for the stacking model
Stacking in RapidMiner

- Accuracy in this experiment:
  - Naive Bayes: 0.71
  - k-NN: 0.81
  - Ripper: 0.71
- Stacked model: 0.86
Stacking

• Variant: also keep the original attributes
• Predictions of base learners are additional attributes for the stacking predictor
  – allows the identification of “blind spots” of individual base learners

• Variant: stacking with confidence values
  – if learners output confidence values, those can be used by the stacking learner
  – often further improves the results
The Classifier Selection Problem

- Question: decision trees or rule learner – which one is better?
- Two corner cases – recap from Data Mining 1

Accuracy:
- Baseline: 0.45
- Decision Tree: 0.45
- Rule Learner: 0.7

- Voting: 0.65
- Weighted Voting: 0.7
- Stacking: 0.83

Accuracy:
- Baseline: 0.89
- Decision Tree: 1.0
- Rule Learner: 0.89

- Voting: 0.89
- Weighted Voting: 1.0
- Stacking: 1.0
Regression Ensembles

• Most ensemble methods also work for regression
  – voting: use average
  – bagging: use average or weighted average
  – stacking: learn *regression* model as stacking model!
  – boosting: the regression variant is called *additive regression*
Additive Regression

• Boosting can be seen as a greedy algorithm for fitting additive models

• Same kind of algorithm for numeric prediction:
  – Build standard regression model
  – Gather residuals, learn model predicting residuals, and repeat
    • Given a prediction \( p(x) \), residual = \( (x-p(x))^2 \)

• To predict, simply sum up weighted individual predictions from all models
Additive Regression w/ Linear Regression

• What happens if we use Linear Regression inside of Additive Regression?

• The first iteration learns a linear regression model $\text{lr}_1$
  – By minimizing the sum of squared errors

• The second iteration aims at learning a LR $\text{lr}_2$ model for
  – $x' = (x - \text{lr}_1(x))^2$
  – Since $(x - \text{lr}_1(x))^2$ is already minimal, $\text{lr}_2$ cannot improve upon this
    • Hence, the subsequent linear models will always be a constant 0
Additive Regression w/ Linear Regression

- First regression model:
Additive Regression w/ Linear Regression

- Second (and third, fourth, ...) regression model:
Additive Regression

- Bottom line: additive and linear regression are not a good match
Example 1: One-dimensional, Non-linear

Linear Regression: RMSE = 0.199

Isotonic Regression: RMSE = 0.171

Additive Isotonic Regression: RMSE = 0.073
Example 2: Multidimensional, Non-Linear

- \( z = 10x^2 - y^3 \)

RMSE of...
- Linear Regression: 385
- Isotonic Regression: 293
- Additive Isotonic Regression: 122
XGBoost

- Currently wins most Kaggle competitions etc.
- Additive Regression w/ Regression Trees
- Regularization
  - Respect size of trees
  - Larger trees: more likely to overfit!
    - Introduce penalty for tree size
  - Overcomes the problem of overfitting in boosting
Intermediate Recap

• Ensemble methods
  – outperform base learners
  – Help minimizing shortcomings of single learners/models
  – simple and complex methods for method combination

• Reasons for performance improvements
  – individual errors of single learners can be “outvoted”
  – more complex hypothesis space
Ensembles for Other Problems

• There are ensembles also for...
• ...clustering (Vega-Pons and Ruiz-Shulkloper, 2011)
  – trying to unify different clusterings
  – using a consensus function mapping different clusterings to each other
• ...outlier detection (Zimek et al., 2014)
  – unifying outlier scores of different approaches
  – requires score normalization and/or rank aggregation
• etc.
Learning with Costs

• Most classifiers aim at reducing the number of errors
  – all errors are regarded as being equally important

• In reality, misclassification costs may differ
• Consider a warning system in an airplane
  – issue a warning if stall is likely to occur
  – based on a classifier using different sensor data
  – wrong warnings may be ignored by the pilot
  – missing warnings may cause the plane to crash

• Here, we have different costs for
  – actual: true, predicted: false → very expensive
  – actual: false, predicted true → not so expensive

http://i.telegraph.co.uk/multimedia/archive/01419/plane_1419831c.jpg
The MetaCost Algorithm

- Form multiple bootstrap replicates of the training set
  - Learn a classifier on each training set
  - i.e., perform bagging
- Estimate each class’s probability for each example
  - by the fraction of votes that it receives from the ensemble
- Use conditional risk equation to relabel each training example
  - with the estimated optimal class
- Reapply the classifier to the relabeled training set
MetaCost

- Conditional risk $R(i|x)$ is the expected cost of predicting that $x$ belongs to class $i$
  - $R(i|x) = \sum P(j|x) C(i, j)$
  - $C(i,j)$ are the classification costs
    (classify an example of class $i$ as class $j$)
  - $P(j|x)$ are obtained by running the bagged classifiers
- The goal of MetaCost procedure is: to relabel the training examples with their “optimal” classes
  - i.e., those with lowest risk
- Then, re-run the classifier to build a final model
  - the resulting classifier will be defensive,
    i.e., make low-risk predictions
  - in the end, the costs are minimized
MetaCost

- **Pilot stall alarm example**
  - $x_1$: stall, $P(stall|x_1) = 0.8$
  - $x_2$: no, $P(no|x_2) = 0.9$

- **Risk values:**
  - $R(stall|x_1) = P(stall|x_1)C(stall,stall) + P(no|x_1)C(stall,no) = 0.2*10 = 20$
  - $R(no|x_1) = P(no|x_1)C(no,stall) + P(no|x_1)C(no,no) = 0.8*1 = 0.8$
  - $R(stall|x_2) = P(stall|x_2)C(stall,stall) + P(no|x_2)C(stall,no) = 0.9*10 = 9$
  - $R(no|x_2) = P(no|x_2)C(stall,no) + P(no|x_2)C(no,no) = 0.1*10 = 1$

- Since $9 > 1$
  - $x_2$ is relabeled to “stall”

<table>
<thead>
<tr>
<th>actual</th>
<th>predicted</th>
<th>stall</th>
<th>no stall</th>
</tr>
</thead>
<tbody>
<tr>
<td>stall</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>no stall</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

8/10 classifiers are correct

http://i.telegraph.co.uk/multimedia/archive/01419/plane_1419831c.jpg
MetaCost vs. Balancing

• Recap balancing:
  – in an unbalanced dataset, there is a bias towards the larger class
  – balancing the dataset helps building more meaningful models

• MetaCost:
  – incidentally unbalance the dataset,
    labeling more instances with the “cheap” class
  – make the learner have a bias towards the “cheap” class
    • i.e., expensive mis-classifications are avoided
  – in the end, the overall cost is reduced

• In the example:
  – there will be more false alarms (stall warning, but actually no stall)
  – the risk of not issuing a warning is reduced
MetaCost in RapidMiner

- Hint: use the performance (cost) operator for evaluation
MetaCost in RapidMiner

- Experiment: set misclassification cost
  Rock $\rightarrow$ Mine = 2; Mine $\rightarrow$ Rock = 1

- Non-cost sensitive decision tree:
  - misclassification cost = 0.33

- MetaCost with decision tree:
  - misclassification cost = 0.24
Another Example for Cost-Sensitive Prediction

• Predicting *ordinal* attributes
  – e.g., very low, low, medium, high, very high

• A standard classifier just looks at correct/incorrect classifications
  – i.e., for a *very low* instance, predicting *low* or *very high* is equally bad

• In practice, predicting *low* for a *very low* instance is much better than predicting *very high*

• Solution: assign costs $C(\text{actual, predicted})$ to predictions
  – $C(\text{very low, very low}) = 0$, $C(\text{very low, low}) = 1$, $C(\text{very low, medium}) = 2$
    ...

Wrap-up

• Ensemble methods in general
  – build a strong model from several weak ones

• Ingredients
  – base learners
  – a combination method

• Variants
  – Voting
  – Bagging (based on sampling)
  – Boosting (based on reweighting instances)
  – Stacking (use learner for combination)

• Also used for cost-sensitive predictions (MetaCost)
Questions?