Admin

• Add/drop deadline is today.
  – You should know by the end of today (tomorrow?) if you’re in the course.
  – As of last night, 20 people left on the waitlist.

• Assignment 1:
  – Due tonight.
  – Late submissions not accepted (so commit/push often!).

• Assignment 2:
  – Coming soon.
  – Specify your partnerships in advance.
Last Time: E-mail Spam Filtering

• Want to build a system that filters spam e-mails:

• We formulated as **supervised learning**:
  – \((y_i = 1)\) if e-mail ‘i’ is spam, \((y_i = 0)\) if e-mail is not spam.
  – \((x_{ij} = 1)\) if word/phrase ‘j’ is in e-mail ‘i’, \((x_{ij} = 0)\) if it is not.

<table>
<thead>
<tr>
<th>$</th>
<th>Hi</th>
<th>CPSC</th>
<th>340</th>
<th>Vicodin</th>
<th>Offer</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>
| ...| ...| ...  | ... | ...     | ...   | ...

Spam?

- 1
- 1
- 0
- ...

...
Last Time: Naïve Bayes

- We considered spam filtering methods based on naïve Bayes:

\[
\Pr(y_i = \text{"spam" } | x_i) = \frac{\Pr(x_i | y_i = \text{"spam"}) \Pr(y_i = \text{"spam"})}{\Pr(x_i)}
\]

- Makes conditional independence assumption to make learning practical:

\[
\Pr(\text{hello, vicodin, CPSC 340 } | \text{spam}) \approx \Pr(\text{hello } | \text{spam}) \Pr(\text{vicodin } | \text{spam}) \Pr(\text{CPSC 340 } | \text{spam})
\]

  - HARD

  - easy

- Predict “spam” if \( \Pr(y_i = \text{"spam" } | x_i) > \Pr(y_i = \text{“not spam” } | x_i) \).
  - We don’t need \( \Pr(x_i) \) to test this.
Decision Trees vs. Naïve Bayes vs. KNN

\[
p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase}) \\
\approx p(\text{milk} \mid \text{sick}) p(\text{egg} \mid \text{sick}) p(\text{lactase} \mid \text{sick}) p(\text{sick})
\]

\[
(\text{milk} = 0.6, \text{egg} = 2, \text{lactase} = 0, ?) \text{ is close to} \\
(\text{milk} = 0.7, \text{egg} = 2, \text{lactase} = 0, \text{sick}) \text{ so predict sick.}
\]
Application: Body-Part Recognition

• Microsoft Kinect:
  – Real-time recognition of 31 body parts from laser depth data.
• How could we write a program to do this?
Supervised Learning Step

• ALL steps are important, but we’ll focus on the learning step.

• Do we have any classifiers that are accurate and run in real time?
  – Decision trees and naïve Bayes are fast, but often not very accurate.
  – KNN is often accurate, but not very fast.

• Deployed system uses an ensemble method called random forests.
Ensemble Methods

• Ensemble methods are classifiers that have classifiers as input.
  – Also called “meta-learning”.

• They have the best names:
  – Averaging.
  – Boosting.
  – Bootstrapping.
  – Bagging.
  – Cascading.
  – Random Forests.
  – Stacking.

• Ensemble methods often have higher accuracy than input classifiers.
Ensemble Methods

• Remember the fundamental trade-off:
  1. $E_{\text{train}}$: How small you can make the training error.
  vs.
  2. $E_{\text{approx}}$: How well training error approximates the test error.

• Goal of ensemble methods is that meta-classifier:
  – Does much better on one of these than individual classifiers.
  – Doesn’t do too much worse on the other.

• This suggests two types of ensemble methods:
  1. **Boosting**: improves training error of classifiers with high $E_{\text{train}}$.
  2. **Averaging**: improves approximation error of classifiers with high $E_{\text{approx}}$. 
Averaging

- Consider a set of classifiers that make these predictions:
  - Classifier 1: “spam”.
  - Classifier 2: “spam”.
  - Classifier 3: “spam”.
  - Classifier 4: “not spam”.
  - Classifier 5: “spam”.
  - Classifier 6: “not spam”.
  - Classifier 7: “spam”.
  - Classifier 8: “spam”.
  - Classifier 9: “spam”.
  - Classifier 10: “spam”.

- If all of these are 80% accurate, what should we predict?
Averaging

• Input to **averaging** is the predictions of a set of models:
  – Decision trees make one prediction.
  – Naïve Bayes makes another prediction.
  – KNN makes another prediction.

• Simple **model averaging**:
  – Take the **mode of the predictions** (or average if probabilistic).
Averaging

• Input to averaging is the predictions of a set of models:
  – Decision trees make one prediction.
  – Naïve Bayes makes another prediction.
  – KNN makes another prediction.

• Stacking:
  – Fit another classifier that uses the predictions as features.
Averaging

• Averaging often performs better than individual models:
  – Averaging typically used by Kaggle winners.
  – E.g., Netflix $1M user-rating competition winner was stacked classifier.

• Why does this work?
• Consider classifiers that tend to overfit (like deep decision trees):
  – If they all overfit in exactly the same way, averaging does nothing.
• But if they make independent errors:
  – Probability of error of average can be lower than individual classifiers.
  – Less attention to specific overfitting of each classifier.
Why does averaging work?

- Consider the models A, B, C applied to training examples 1,2,3.
- The models make different errors, so averaging improves accuracy.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Averaged model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✔ ✔ ✗ ✔ ➔ ✔</td>
</tr>
<tr>
<td>2</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔ ✗ ✔ ✔ ➔ ✔</td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✗ ✔ ✔ ✔ ➔ ✔</td>
</tr>
</tbody>
</table>
Random Forests

• Random forests **average a set of deep decision trees.**
  – Tend to **be one of the best “out of the box” classifiers.**
    • Often close to the best performance of any method on the first run.
  – And **predictions are very fast.**

• Do deep decision trees make independent errors?
  – No: with the same training data you’ll get the same decision tree.

• Two key ingredients in random forests:
  – **Bootstrapping.**
  – **Random trees.**
Random Forest Ingredient 1: Bootstrap

• **Bootstrap sample** of a list of ‘n’ objects:
  – A set of ‘n’ objects chosen independently with replacement.
    
    ```
    for i in 1:n  
    j = rand(1:n)  # pick a random number from 1, 2, ..., n  
    X_{bootstrap}[i] = X[j]  # use the random sample
    ```

  – Gives new dataset of ‘n’ objects, with some duplicated and some missing.
    • Approximately 63% of original objects will be included for large ‘n’.
    – Very common in statistics to estimate sensitivity of statistic to data.

• **Bagging**: using bootstrap samples for ensemble learning.
  – Generate several bootstrap samples of the objects \((x_i, y_i)\).
  – Fit a classifier to each bootstrap sample.
  – At test time, average the predictions.
Random Forest Ingredient 2: Random Trees

- For each split in a random tree model:
  - Randomly sample a small number of possible features.
  - Only consider these random features when searching for the optimal rule.

Random tree 1:
- Sample (milk, oranges)  $\text{milk} > 0.5$

Random tree 2:
- Sample (eggy lactase) $\text{egg} > 0$
Random Forest Ingredient 2: Random Trees

• For each split in a random tree model:
  – Randomly sample a small number of possible features.
  – Only consider these random features when searching for the optimal rule.
Random Forest Ingredient 2: Random Trees

• For each split in a random tree model:
  – Randomly sample a small number of possible features.
  – Only consider these random features when searching for the optimal rule.

• Splits will tend to use different features in different trees.
  – They will still overfit, but hopefully make *independent* errors.

• So the average tends to have a much lower test error.

• Empirically, random forests are one of the “best” classifiers.

• Fernandez-Delgado et al. [2014]:
  – Compared 179 classifiers on 121 datasets.
  – Random forests are most likely to be the best classifier.
Random Forests

• Random forests are one of the best ‘out of the box’ classifiers.
• Fit deep decision trees to random bootstrap samples of data, base splits on random subsets of the features, and classify using mode.
Random Forests

- Random forests are one of the best ‘out of the box’ classifiers.
- Fit deep decision trees to random bootstrap samples of data, base splits on random subsets of the features, and classify using mode.
End of Part 1: Key Concepts

• Fundamental ideas:
  – Training vs. test error (memorization vs. learning).
  – IID assumption (examples come independently from same distribution).
  – Golden rule of ML (test set should not influence training).
  – Fundamental trade-off (between training error vs. approximation error).
  – Validation sets and cross-validation (can approximate test error).
  – Optimization bias (we can overfit the training set and the validation set).
  – Decision theory (we should consider costs of predictions).
  – Parametric vs. non-parametric (whether model size depends on ‘n’).
  – No free lunch theorem (there is no “best” model).
End of Part 1: Key Concepts

• We saw 3 ways of “learning”:
  – Searching for rules.
    • Decision trees (greedy recursive splitting using decision stumps).
  – Counting frequencies.
    • Naïve Bayes (probabilistic classifier based on conditional independence).
  – Measuring distances.
    • K-nearest neighbours (non-parametric classifier with universal consistency).

• We saw 2 generic ways of improving performance:
  – Encouraging invariances with data augmentation.
  – Ensemble methods (combine predictions of several models).
    • Random forests (averaging plus randomization to reduce overfitting).
Summary

• **Ensemble methods** take classifiers as inputs.
  • Try to reduce either $E_{\text{train}}$ or $E_{\text{approx}}$ without increasing the other much.

• **Averaging:**
  • Improves predictions of multiple classifiers if errors are independent.

• **Random forests:**
  • Averaging of deep randomized decision trees.
  • One of the best “out of the box” classifiers.

• Next time:
  • We start unsupervised learning.
Some Ingredients of Kinect

1. Collect **hundreds of thousands of labeled images** (motion capture).
   - Variety of pose, age, shape, clothing, and crop.
2. Build a **simulator that fills space of images** by making even more images.
3. Extract **features of each location**, that are cheap enough for real-time calculation (depth differences between pixel and pixels nearby.)
4. Treat classifying body part of a pixel as a supervised learning problem.
5. Run classifier in parallel on all pixels using graphical processing unit (GPU).

Why does Bootstrapping select approximately 63%?

• Probability of an arbitrary $x_i$ being selected in a bootstrap sample:

$$p(\text{selected at least once in } 'n' \text{ trials})$$

$$= 1 - p(\text{not selected in any of } 'n' \text{ trials})$$

$$= 1 - (1 - 1/n)^n$$

$$\approx 1 - 1/e$$

$$\approx 0.63$$
Why can Averaging Work?

• Consider having ‘3’ binary classifiers, that are each independently right with probability 0.80:
  –  \( P(\text{all 3 right}) = 0.8^3 = 0.512 \).
  –  \( P(2 \text{ rights, 1 wrong}) = 3 \times 0.8^2(1-0.8) = 0.384 \).
  –  \( P(1 \text{ right, 2 wrongs}) = 3 \times (1-0.8)^20.8 = 0.096 \).
  –  \( P(\text{all 3 wrong}) = (1-0.8)^3 = 0.008 \).

• So ensemble is right with probability 0.896 (which is 0.512+0.384).
  – Note that it’s important that classifiers are at least somewhat independent, have probability of being right > 0.5, and that the probabilities aren’t too different (otherwise, you may be better off just picking the best one).
Bonus Slide: Why Random Forests Work

• Consider ‘k’ independent classifiers, whose errors have a variance of $\sigma^2$.
• If the errors are IID, the variance of the average is $\sigma^2/k$.
  – So the more classifiers you average, the more you decrease error variance.
  (And the more the training error approximates the test error.)
• Generalization to case where classifiers are not independent is:
  
  $$c \sigma^2 + \left(1 - c \right) \frac{\sigma^2}{k}$$

  – Where ‘c’ is the correlation.
• So the less correlation you have the closer you get to independent case.
• Randomization in random forests decreases correlation between trees.
  – See also “Sensitivity of Independence Assumptions”.

28
Boosting: Key Ideas

• Input to boosting is classifier that:
  – Is simple enough that it doesn’t overfit much.
  – Can obtain >50% weighted training accuracy.

• Example: decision stumps or low-depth decision trees.
Boosting: Key Ideas

• Basic steps:
  1. Fit a classifier on the training data.
  2. Give a higher weight to examples that the classifier got wrong.
  3. Fit a classifier on the weighted training data.
  4. Go back to 2.

• Final prediction: weighted vote of individual classifier predictions.

• Boosted decision trees are very fast/accurate classifiers.
  – “XGBoost”: recent method that has been winning Kaggle competitions.
How these concepts often show up in practice

• Here is a recent e-mail related to many ideas we’ve recently covered:
  – “However, the performance did not improve while the model goes deeper and with augmentation. The best result I got on validation set was 80% with LeNet-5 and NO augmentation (LeNet-5 with augmentation I got 79.15%), and later 16 and 50 layer structures both got 70%~75% accuracy.

  In addition, there was a software that can use mathematical equations to extract numerical information for me, so I trained the same dataset with nearly 100 features on random forest with 500 trees. The accuracy was 90% on validation set.

  I really don't understand that how could deep learning perform worse as the number of hidden layers increases, in addition to that I have changed from VGG to ResNet, which are theoretically trained differently. Moreover, why deep learning algorithm cannot surpass machine learning algorithm?”

• Above there is data augmentation, validation error, effect of the fundamental trade-off, the no free lunch theorem, and the effectiveness of random forests.
Bonus Slide: Bayesian Model Averaging

• Recall the key observation regarding ensemble methods:
  – If models overfit in “different” ways, averaging gives better performance.

• But should all models get equal weight?
  – E.g., decision trees of different depths, when lower depths have low training error.
  – E.g., a random forest where one tree does very well (on validation error) and others do horribly.
  – In science, research may be fraudulent or not based on evidence.

• In these cases, naïve averaging may do worse.
Suppose we have a set of ‘m’ probabilistic binary classifiers $w_j$.

If each one gets equal weight, then we predict using:

$$p(y_i | x_i) = \frac{1}{m} p(y_i | w_1, x_i) + \frac{1}{m} p(y_i | w_2, x_i) + \cdots + \frac{1}{m} p(y_i | w_m, x_i)$$

**Bayesian model averaging** treats model ‘$w_j$’ as a random variable:

$$p(y_i | x_i) = \mathbb{E}_{w_j \perp x_i} p(y_i \mid w_j, x_i) = \sum_{j=1}^{m} p(y_i \mid w_j, x_i) p(w_j \mid x_i) = \sum_{j=1}^{m} p(y_i | w_j, x_i) p(w_j)$$

So we should weight by probability that $w_j$ is the correct model:

- Equal weights assume all models are equally probable.
Bonus Slide: Bayesian Model Averaging

• Can get better weights by conditioning on training set:

\[
p(w_j | X, y) \propto p(y | w_j, X) p(w_j | X) = p(y | w_j, X) p(w_j)
\]

• The ‘likelihood’ \( p(y | w_j, X) \) makes sense:
  – We should give more weight to models that predict ‘y’ well.
  – Note that hidden denominator penalizes complex models.

• The ‘prior’ \( p(w_j) \) is our ‘belief’ that \( w_j \) is the correct model.

• This is how rules of probability say we should weigh models.
  – The ‘correct’ way to predict given what we know.
  – But it makes some people unhappy because it is subjective.