Today’s quiz

- Consider the training data below (marked “+”, “-” for the output), and the test data (marked “A”, “B”, “C”).
  - What class would the 1-NN algorithm predict for each test point?
  - What class would the 3-NN algorithm predict for each test point?
  - Which training points could be removed without causing change in the 3-NN decision rule?

- Properties of NN algorithm?

```
+  +  ●A  -  -
+  +  +  -  -
+  +  -  ●B  -
+  ●C  -  -  -
```
Mini-project #1 reviewing

- Still missing 45 reviews (out of 80). You can still submit and edit your review until tonight.

- Most are very well done, on-topic, constructive, detailed.

- Some only highlight the qualities. Any suggestions for improvements?

- Some only provide a list of suggestions. It’s good to highlight what are the strengths of the project.

Constructive:

(...)
Quality and thoroughness of presentation and analysis of the results:

Presentation format was very good and included many figures and graphs as visual aids in order to allow the reader to have a more concrete sense of the material being presented. There were many grammatical errors, and the overall sentence structure was slightly poor. In terms of analysis of the results, the flow was logical and smooth. Too much detail about weight analysis, and not enough material about testing error and how it relates to methods such as normalization of features and labels, or the relationship between training error and testing error.

(...)
Specific and detailed:

(...) In section II, I like that the assumption about linear regression is clearly stated. On the other hand, in equation 1, I think the $w_i$ and $x_i$ should have been better explained. In section II-A, it is not clear for me which process was repeated 50 times (end of first paragraph). At the end of that section, I'm not sure why you are doing cross-validation: which parameter are you trying to optimize? Also, are you doing cross-validation on the test set?

For section II-B, I think the $x_{ij}$ should be explained in equation 3. In the third paragraph, consistence should be use when refering to alphas (you used alphas and then the symbol for it). In the fourth paragraph, how did you choose to multiply alpha by 0.085? Similarly, in the sixth paragraph, how did you choose to multiply alpha by 0.72?

(...)  

Pedagogical:

(...)  

Quality of methodology, implementation of algorithms: It seems to me that you have misunderstood how cross-validation works. There is no need to cross-validate for multiple values of k (in k-fold cross-validation). One should select a value of k to cross-validate over the selected features or an algorithm's meta-parameters. Moreover, the test set should never be used in the cross-validation process. It should be left out until the very end.

(...)
Going the extra mile:

(...) 

Your programming is good, but I cannot find the "sports.csv" of your codes in either the zip file or the GitHub link you provided, so that I am not able to run the codes to identify your result. 

(...) 

Philosophical:

(...) 

The code was messy, but then again it appears like most MATLAB code is very messy. 

(...)
Mini-project #2

- 8+1 submissions to Kaggle. Max. number of submissions per day bumped to 3.

- Best score currently 61.8%.

- A note on using TF-IDF (and other features):
  - You can use an outside corpus to evaluate the weights if you want.

Next topic: Ensemble methods

- Recently seen supervised learning methods:
  - Logistic regression, Naïve Bayes, LDA/QDA
  - Decision trees, Instance-based learning

- Core idea of decision trees? Build complex classifiers from simpler ones.
  E.g. Linear separator -> Decision trees

- Several ways to do this.
  - Bagging
  - Random forests
  - Boosting
Ensemble learning in general

- **Key idea:** Run a base learning algorithm multiple times, then combine the predictions of the different learners to get a final prediction.
  - What’s a base learning algorithm?
    - Naïve Bayes, LDA, Decision trees, SVMs, ...

- **First attempt:** Construct several classifiers independently.
  - Bagging.
  - Randomizing the test selection in decision trees (Random forests).
  - Using a different subset of input features to train different trees.

- **More complex approach:** Coordinate the construction of the hypotheses in the ensemble.

---

Recall bootstrapping

- Given dataset $D$, construct a bootstrap replicate of $D$, called $D_k$, which has the same number of examples, by drawing samples from $D$ with replacement.
- Use the learning algorithm to construct a hypothesis $h_k$ by training on $D_k$.
- Compute the prediction of $h_k$ on each of the remaining points, from the set $T_k = D - D_k$.
- Repeat this process $K$ times, where $K$ is typically a few hundred.
- If $D$ is very large, the replicates can contain $n < |D|$ points (still drawn with replacement.)
Estimating bias and variance

- For each point \( x \), we have a set of estimates \( h_1(x), \ldots, h_K(x) \), with \( K \leq B \) (since \( x \) might not appear in some replicates).
- The average empirical prediction of \( x \) is: \( \hat{h}(x) = \frac{1}{K} \sum_{k=1}^{K} h_k(x) \).
- We estimate the bias as: \( y - \hat{h}(x) \).
- We estimate the variance as: \( \frac{1}{K-1} \sum_{k=1}^{K} (\hat{h}(x) - h_k(x))^2 \).

Bagging: Bootstrap aggregation

- If we did all the work to get the hypotheses \( h_b \), why not use all of them to make a prediction? (as opposed to just estimating bias/variance/error).
- All hypotheses get to have a vote.
  - For classification: pick the majority class.
  - For regression, average all the predictions.
- Which hypotheses classes would benefit most from this approach?
Bagging

- For each point \( x \), we have a set of estimates \( h_1(x), \ldots, h_K(x) \), with \( K \leq B \) (since \( x \) might not appear in some replicates).
- The average empirical prediction of \( x \) is: \( \hat{h}(x) = \frac{1}{K} \sum_{k=1}^{K} h_k(x) \).
- We estimate the bias as: \( y - \hat{h}(x) \).
- We estimate the variance as: \( \frac{1}{(K-1)} \sum_{k=1}^{K} (\hat{h}(x) - h_k(x))^2 \).
- In theory, bagging eliminates variance altogether.
- In practice, bagging tends to reduce variance and increase bias.
- Use this with "unstable" learners that have high variance, e.g. decision trees, neural networks, nearest-neighbour.

Random forests (Breiman)

- Basic algorithm:
  - Use \( K \) bootstrap replicates to train \( K \) different trees.
  - At each node, pick \( m \) variables at random (use \( m < M \), the total number of features).
  - Determine the best test (using normalized information gain).
  - Recurse until the tree reaches maximum depth (no pruning).
- Comments:
  - Each tree has high variance, but the ensemble uses averaging, which reduces variance.
  - Random forests are very competitive in both classification and regression, but still subject to overfitting.
Extremely randomized trees (Geurts et al., 2005)

• Basic algorithm:
  – Construct $K$ decision trees.
  – Pick $m$ attributes at random (without replacement) and pick a random test involving each attribute.
  – Evaluate all tests (using a normalized information gain metric) and pick the best one for the node.
  – Continue until a desired depth or a desired number of instances ($n_{\text{min}}$) at the leaf is reached.

• Comments:
  – Very reliable method for both classification and regression.
  – The smaller $m$ is, the more randomized the trees are; small $m$ is best, especially with large levels of noise. Small $n_{\text{min}}$ means less bias and more variance, but variance is controlled by averaging over trees.

Randomization in general

• Instead of searching very hard for the best hypothesis, generate lots of random ones, then average their results.

• Examples:

• Advantages?
  – Very fast, easy, can handle lots of data.
  – Can circumvent difficulties in optimization.
  – Averaging reduces the variance introduced by randomization.

• Disadvantages?
  – New prediction may be more expensive to evaluate (go over all trees).
  – Still typically subject to overfitting.
  – Low interpretability compared to standard decision trees.
Additive models

• In an ensemble, the output on any instance is computed by averaging the outputs of several hypotheses.

• **Idea:** Don’t construct the hypotheses independently. Instead, new hypotheses should focus on instances that are problematic for existing hypotheses.
  – If an example is difficult, more components should focus on it.

Boosting

Recall algorithm:
– Use the training set to train a simple predictor.
– Re-weight the training examples, putting more weight on examples that were not properly classified in the previous predictor.
– Repeat $n$ times.
– Combine the simple hypotheses into a single, accurate predictor.
Notation

• Assume that examples are drawn independently from some probability distribution $P$ on the set of possible data $D$.

• Let $J_P(h)$ be the expected error of hypothesis $h$ when data is drawn from $P$:

$$J_P(h) = \sum_{x,y} J(h(x),y)P(<x,y>)$$

where $J(h(x),y)$ could be the squared error, or 0/1 loss.

Weak learners

• Assume we have some "weak" binary classifiers:
  – A decision stump is a single node decision tree: $x > t$
  – A single feature Naïve Bayes classifier.
  – A 1-nearest neighbour classifier.

• “Weak” means $J_P(h) < \frac{1}{2} - \gamma$ (assuming 2 classes), where $\gamma > 0$
  – So true error of the classifier is only slightly better than random.

• Questions:
  – How do we re-weight the examples?
  – How do we combine many simple predictors into a single classifier?
Example

Example: First step

\[ D_1 \]

\[ D_2 \]

\[ h_1 \]

\( \epsilon_1 = 0.30 \)

\( \alpha_1 = 0.42 \)
Example: Second step

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

Example: Third step

\[ \varepsilon_3 = 0.14 \]
\[ \alpha_3 = 0.92 \]
Example: Final hypothesis

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

AdaBoost (Freund & Schapire, 1995)

Given: \( (x_1, y_1), \ldots, (x_m, y_m) \) where \( x_i \in X, y_i \in Y = \{-1, +1\} \)
Initialize \( D_1(i) = \frac{1}{m} \).
For \( t = 1, \ldots, T \):
- Train weak learner using distribution \( D_t \).
- Get weak hypothesis \( h_t : X \rightarrow \{-1, +1\} \) with error
  \[ \epsilon_t = \Pr_{x \sim D_t} [h_t(x) \neq y_t] \]
- Choose \( \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \). weight of weak learner \( t \)
- Update:
  \[ D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \]
  \[ = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \]
  where \( Z_t \) is a normalization factor (chosen so that \( D_{t+1} \) will be a distribution).

Output the final hypothesis:
\[ H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) \]
Properties of AdaBoost

- Compared to other boosting algorithms, main insight is to automatically adapt the error rate at each iteration.

- Training error on the final hypothesis is at most:
  \[
  \prod_t \left[ 2^{\sqrt{r_t(1 - \epsilon_t)}} \right] = \prod_t \sqrt{1 - 4 \gamma_t^2} \leq \exp \left( -2 \sum_t \gamma_t^2 \right)
  \]
  recall: \( \gamma_t \) is how much better than random is \( h_t \)

- AdaBoost gradually reduces the training error exponentially fast.

---

Real data set: Text categorization

![Graphs showing performance of AdaBoost on different datasets](image)

- database: AP
- database: Reuters
3.4 RESULTS AND DISCUSSION

One of our goals in carrying out these experiments was to determine if boosting using pseudo-loss (rather than error) is worthwhile. Figure 3 shows how the different algorithms performed on the benchmarks. For either error or pseudo-loss, the differences between bagging and boosting were not statistically significant, with bagging generally performing slightly better.

Figure 4 shows similar scatterplots comparing the performance of boosting and bagging for all the benchmarks. The boosting error rates were slightly lower than the bagging error rates for most benchmarks, but the differences were not always statistically significant.

Figure 5 shows a similar comparison for boosting and bagging with error. For error, boosting performed significantly better than bagging for most benchmarks. However, for pseudo-loss, the differences were not statistically significant, with bagging performing slightly better in most cases.

In summary, bagging and boosting both had their strengths and weaknesses, with bagging generally performing slightly better for error and boosting performing slightly better for pseudo-loss. The differences were not always statistically significant, but the trends were clear.
Bagging vs Boosting

- Bagging is typically faster, but may get a smaller error reduction (not by much).
- Bagging works well with “reasonable” classifiers.
- Boosting works with very simple classifiers.
  E.g., Boostexter - text classification using decision stumps based on single words.
- Boosting may have a problem if a lot of the data is mislabeled, because it will focus on those examples a lot, leading to overfitting.

Why does boosting work?

- Weak learners have high bias. By combining them, we get more expressive classifiers. Hence, boosting is a bias-reduction technique.
- Adaboost looks for a good approximation to the log-odds ratio, within the space of functions that can be captured by a linear combination of the base classifiers.
- What happens as we run boosting longer? Intuitively, we get more and more complex hypotheses. How would you expect bias and variance to evolve over time?
A naïve (but reasonable) analysis of error

- Expect the training error to continue to drop (until it reaches 0).
- Expect the test error to increase as we get more voters, and $h_t$ becomes too complex.

![Graph showing training and test error over iterations.]

-------

Actual typical run of AdaBoost

- Test error does not increase even after 1000 runs! (more than 2 million decision nodes!)
- Test error continues to drop even after training error reaches 0!
- These are consistent results through many sets of experiments!
- Conjecture: Boosting does not overfit!
What you should know

• Ensemble methods combine several hypotheses into one prediction.
• They work better than the best individual hypothesis from the same class because they reduce bias or variance (or both).
• Extremely randomized trees are a bias-reduction technique.
• Bagging is mainly a variance-reduction technique, useful for complex hypotheses.
• Main idea is to sample the data repeatedly, train several classifiers and average their predictions.

• Boosting focuses on harder examples, and gives a weighted vote to the hypotheses.
• Boosting works by reducing bias and increasing classification margin.