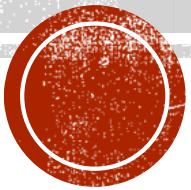
IMBALANCED DATA AND REDUCTIONS



CS 412 Introduction to Machine Learning

Prof. Zheleva

February 8, 2018

Reading Assignment: CIML: 6

LAST LECTURE: PRACTICAL ISSUES

- Garbage in, garbage out
- Importance of feature representation
- Evaluation: beyond accuracy
- Cross validation

ML APPLICATION DESIGN

1	real world goal	increase revenue
2	real world mechanism	better ad display
3	learning problem	classify click-through
4	data collection	interaction w/ current system
5	collected data	query, ad, click
6	data representation	bow^2, $\pm$ click
7	select model family	decision trees, depth 20
8	select training data	subset from april'16
9	train model & hyperparams	final decision tree
10	predict on test data	subset from may'16
11	evaluate error	zero/one loss for $\pm$ click
12	deploy!	(hope we achieve our goal)

IMPROVING INPUT REPRESENTATION

- Feature pruning
- Feature normalization

Centering: $$x_{n,d} \leftarrow x_{n,d} - \mu_d \tag{5.1}$$

Variance Scaling: $$x_{n,d} \leftarrow x_{n,d} / \sigma_d \tag{5.2}$$

Absolute Scaling: $$x_{n,d} \leftarrow x_{n,d} / r_d \tag{5.3}$$

where:

$$\mu_d = \frac{1}{N} \sum_n x_{n,d} \tag{5.4}$$

$$\sigma_d = \sqrt{\frac{1}{N-1} \sum_n (x_{n,d} - \mu_d)^2} \tag{5.5}$$

$$r_d = \max_n \left| x_{n,d} \right| \tag{5.6}$$

- Example normalization

EVALUATION WHEN ONE CLASS IS MORE IMPORTANT

- Precision: % positive predictions that are correct

$$Precision = \frac{TP}{TP + FP}$$

- Recall: % positive true labels that are predicted

$$Recall = \frac{TN}{TP + FP}$$

	True label = +1	**True label = -1**
Predicted label = +1	True positives (TP)	False positives (FP)
True label = -1	False negatives (FN)	True negatives (TN)

TODAY: MORE PRACTICAL ISSUES

- Bias-variance tradeoff
- Dealing with imbalanced data distributions
- How to use binary classifiers for multiclass classification problems
- Algorithms & guarantees on error rate
- Fundamental ML concepts
  - Reductions

CROSS-VALIDATION

Algorithm 8 CROSSVALIDATE(*LearningAlgorithm*, *Data*, K)

```
1:  ê ← ∞                                          // store lowest error encountered so far
2:  α̂ ← unknown                                    // store the hyperparameter setting that yielded it
3:  for all hyperparameter settings α do
4:      err ← [ ]                                  // keep track of the K-many error estimates
5:      for k = 1 to K do
6:          train ← {(x_n, y_n) ∈ Data : n mod K ≠ k − 1}
7:          test ← {(x_n, y_n) ∈ Data : n mod K = k − 1}   // test every Kth example
8:          model ← Run LearningAlgorithm on train
9:          err ← err ⊕ error of model on test     // add current error to list of errors
10:     end for
11:     avgErr ← mean of set err
12:     if avgErr < ê then
13:         ê ← avgErr                             // remember these settings
14:         α̂ ← α                                  // because they're the best so far
15:     end if
16: end for
```

Algorithm 9 KNN-TRAIN-LOO(**D**)

1: $err_k \leftarrow 0, \forall 1 \leq k \leq N-1$ // err_k stores how well you do with kNN
2: **for** $n = 1$ **to** N **do**
3: $S_m \leftarrow \langle ||\boldsymbol{x}_n - \boldsymbol{x}_m||, m \rangle, \forall m \neq n$ // compute distances to other points
4: $S \leftarrow \text{SORT}(S)$ // put lowest-distance objects first
5: $\hat{y} \leftarrow 0$ // current label prediction
6: **for** $k = 1$ **to** $N-1$ **do**
7: $\langle dist, m \rangle \leftarrow S_k$
8: $\hat{y} \leftarrow \hat{y} + y_m$ // let kth closest point vote
9: **if** $\hat{y} \neq y_m$ **then**
10: $err_k \leftarrow err_k + 1$ // one more error for kNN
11: **end if**
12: **end for**
13: **end for**
14: **return** $\text{argmin}_k \, err_k$ // return the K that achieved lowest error

COMPARISON BETWEEN ALGORITHMS

- Is an algorithm with accuracy of 7% better than one with 6.9%?
  - Need a notion of statistical significance

- Null hypotheses: these two algorithms are the same

- Compute average and standard deviation of metric over K folds
  - Assume distribution of scores (e.g., accuracy) is Gaussian
  - If ($mean_1$ -2*$stddev_1$)>($mean_2$ +$stddev_2$), 95% confidence in difference

DEBUGGING

- You've implemented a learning algorithm,
- You try it on some train/dev/test data
- You get really bad performance
- What's going on?
  - Is the data too noisy?
  - Is the learning problem too hard?
  - Is the implementation of the learning algorithm buggy?

STRATEGIES FOR ISOLATING CAUSES OF ERRORS

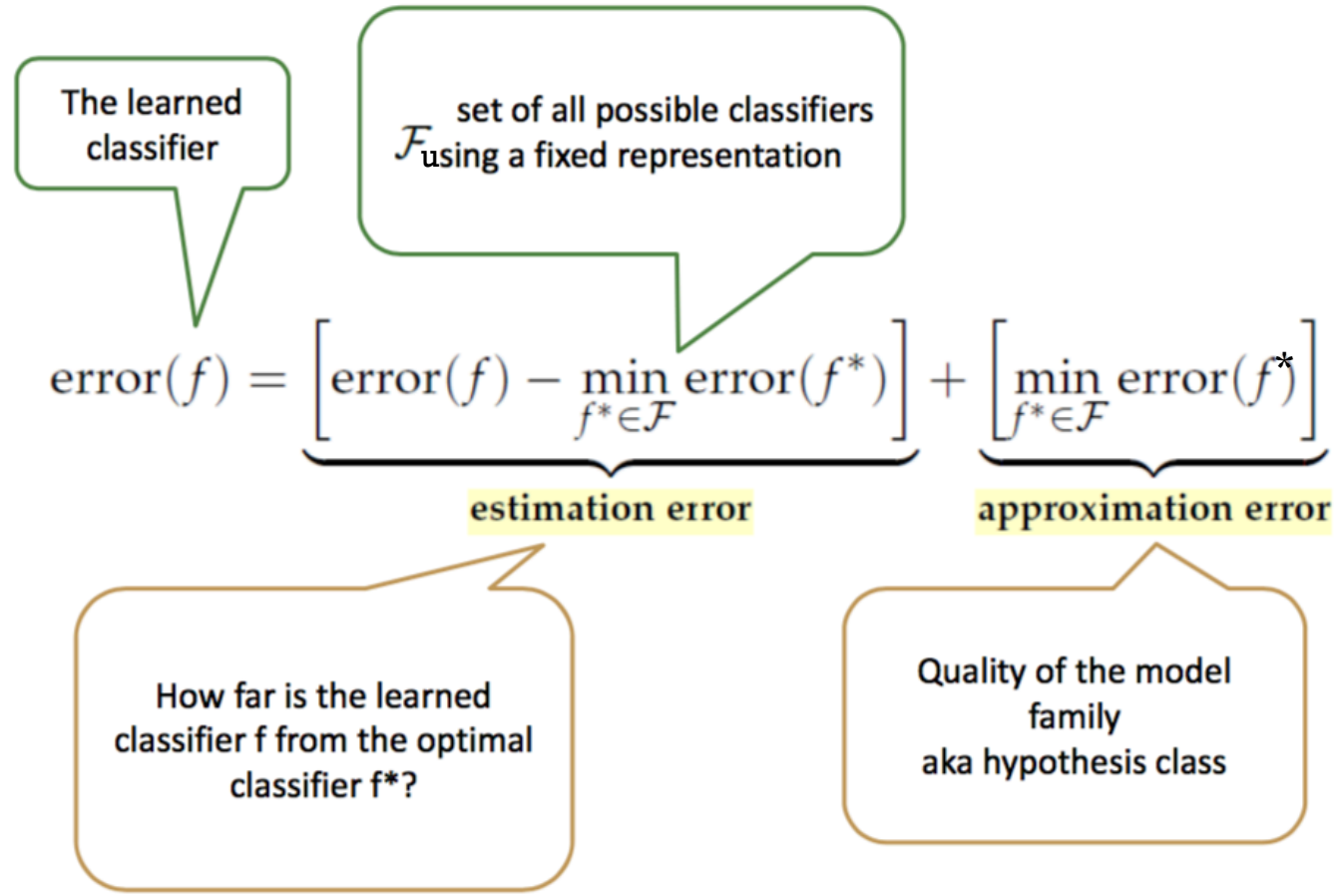
- Is the problem with **generalization** to test data?
  - Can learner fit the training data?
    - Yes: problem is in generalization to test data
    - No: problem is in representation (need better features or better data)
- **Train/test mismatch**?
  - Try reselecting train/test by shuffling training data and test together

STRATEGIES FOR ISOLATING CAUSES OF ERRORS

- Is algorithm **implementation correct**?
  - Hand-craft a toy dataset
- **Is representation adequate?**
  - Can you learn if you add a cheating feature that perfectly correlates with correct class?
- Do you have **enough data**?
  - Try training on 80% of the training set, how much does it hurt performance?

FORMALIZING ERRORS

- Impossible to compute but representation is useful for debugging
  - More complex functions mean smaller appx. error, higher likelihood to overfit



BIAS-VARIANCE TRADEOFF

- Trade-off between
  - approximation error (bias)
  - estimation error (variance)
- Example:
  - Consider the always positive classifier
    - Low variance as a function of a random draw of the training set
    - Strongly biased toward predicting +1 no matter what the input

BIAS AND VARIANCE

- **Expectation**: what should happen <u>on average</u>

$$E[f(X)] = \sum P(x)\, f(x)$$

  - Linearity properties

$$E[f_1(X) + f_2(X)] = E[f_1(X)] + E[f_2(X)]$$

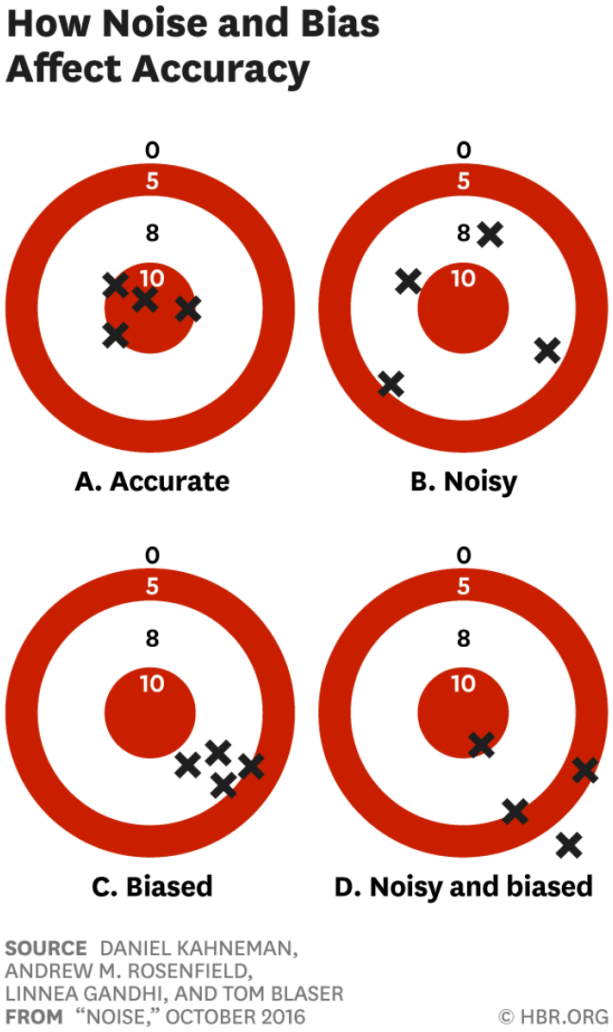
$$E[\alpha\, f(X)] = \alpha\, E[f(X)]$$

- **Bias**: difference between expected and true values

$$\text{Bias}(f_D(x)) = E[f_D(x)] - f(x)$$

- **Variance**: measure of the "spread" of a distribution

$$\text{Var}(f_D(x)) = E[(f_D(x) - E[f_D(x)])^2] = E[f_D(x)^2] - (E[f_D(x)])^2$$



ESTIMATING THE LOSS/ERROR

Assumptions:

$Y = f(X)+\epsilon$

Irreducible error $\epsilon \sim N(0,\sigma_\epsilon^2)$

Loss function/expected error:

$$Err(x) = E\left[(Y - \hat{f}_D(x))^2\right]$$

Bias-Variance Decomposition

$$Err(x) = \left(E[\hat{f}_D(x)] - f(x)\right)^2 + E\left[\left(\hat{f}_D(x) - E[\hat{f}_D(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

BIAS-VARIANCE TRADE-OFF FOR KNN

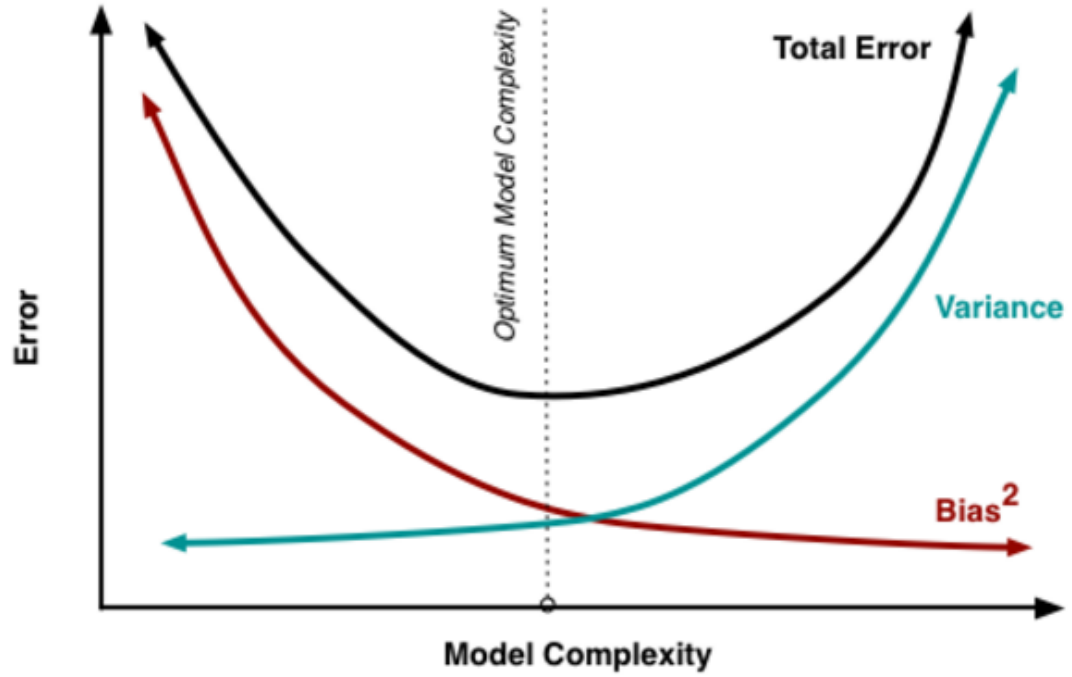
$$Err(x) = \left(E[\hat{f}_D(x)] - f(x)\right)^2 + E\left[\left(\hat{f}_D(x) - E[\hat{f}_D(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Derivation for k-NN: $$Err(x) = \left(f(x) - \frac{1}{k}\sum_{i=1}^{k} f(x_i)\right)^2 + \frac{\sigma_\epsilon^2}{k} + \sigma_\epsilon^2$$

What happens with the bias and variance as k increases?

BIAS-VARIANCE TRADE-OFF

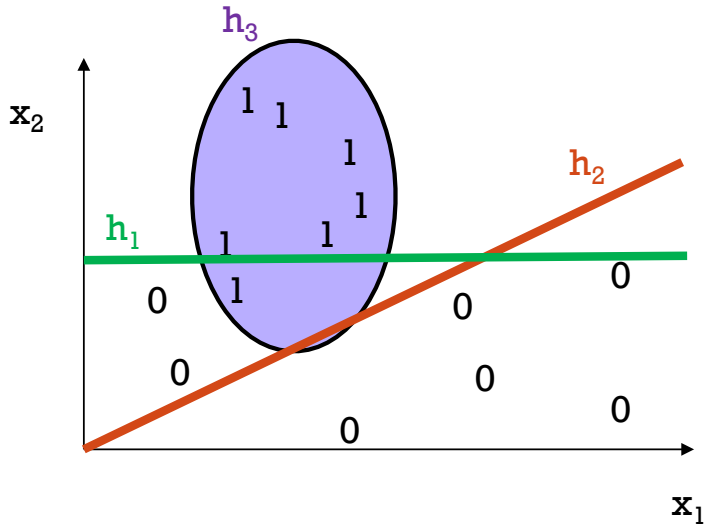


IMBALANCED DATA DISTRIBUTION

- Sometimes training examples are drawn from an imbalanced distribution
  - This results in an imbalanced training set
  - "needle in a haystack" problem
  - E.g., find fraudulent transactions in credit card histories
- Why is this a big problem for the ML algorithms we know?

RECALL: ML AS FUNCTION APPROXIMATION

- Problem setting
  - **X** – set of possible instances
  - Y – target class
  - Unknown target function f: **X** $\rightarrow$ Y
  - Set of function hypotheses H={h | h: **X** $\rightarrow$ Y}



- Input
  - Training examples $\{(\boldsymbol{x}^{(1)}, y^{(1)}),\ldots(\boldsymbol{x}^{(N)}, y^{(N)})\}$ of unknown target function

- Output
  - Hypothesis $h \in$ H that best approximates target function f

RECALL: FORMALIZING INDUCTION

- Loss function: $\text{loss}(y, f(x))$ where y is the true class label and f(x) is the system's prediction
  - Captures our notion of what is important to learn
  - For example, $\text{loss}(y, f(x)) = \begin{cases} 0 \; if \; y = f(x) \\ 1 \quad otherwise \end{cases}$
- Important question: where does the data come from?
  - Data generating probability distribution D over (**x**,y) pairs
  - Typically, we don't know what D is
  - We have a sample from it: our training data

RECALL: FORMALIZING INDUCTION

- f(**x**) should make good predictions
  - As measured by loss function
  - On **future** examples (typically) drawn from D

- Central to learning: minimization of the *expected loss*

$$\varepsilon \triangleq \mathbb{E}_{(x,y)\sim D}\{l(y, f(x))\} = \sum_{(x,y)} D(x,y)\, l(y, f(x))$$

- Approximate expected loss through training error

$$\hat{\varepsilon} \triangleq \sum_{n=1}^{N} \frac{1}{N} l(y^{(n)}, f(x^{(n)}))$$

TASK: BINARY CLASSIFICATION

Given:

1. An input space $\mathcal{X}$
2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$
3. A training set D sampled from $\mathcal{D}$

Compute: A function f minimizing: $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[f(\boldsymbol{x}) \neq y\right]$

TASK: α-WEIGHTED BINARY CLASSIFICATION

Given:

1. An input space $\mathcal{X}$
2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$
3. A training set D sampled from $\mathcal{D}$

Compute: A function f minimizing: $\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\alpha^{y=1}[f(\boldsymbol{x}) \neq y]\right]$

We define cost of misprediction as $\alpha > 1$ for y=+1 and 1 for y=-1

- Given a good algorithm for solving the binary classification problem, how can we solve the α-weighted binary classification problem?

SOLUTION: TRAIN AND EVALUATE A BINARY CLASSIFIER ON AN INDUCED DISTRIBUTION

Algorithm 11 SUBSAMPLEMAP($\mathcal{D}^{weighted}$, α)

1: **while** *true* **do**
2: $(x, y) \sim \mathcal{D}^{weighted}$ // draw an example from the weighted distribution
3: $u \sim$ uniform random variable in $[0, 1]$
4: **if** $y = +1$ **or** $u < \frac{1}{\alpha}$ **then**
5: **return** (x, y)
6: **end if**
7: **end while**

Algorithm 12 SUBSAMPLETEST(f^{BINARY}, $\hat{x}$)

1: **return** $f^{\text{BINARY}}(\hat{x})$

SUBSAMPLING OPTIMALITY

- **Theorem:** If the binary classifier on the subsampled data achieves a binary error rate of ε, then the error rate of the α-weighted classifier is α ε
  - If it does well on the sample, it will do well on the original distribution
  - Assume D^w is the original distribution and D^b the induced one

$$\epsilon^w = \mathbb{E}_{(x,y)\sim\mathcal{D}^w}\left[\alpha^{y=1}\left[f(x) \neq y\right]\right] \tag{6.1}$$

$$= \sum_{x\in\mathcal{X}}\sum_{y\in\pm 1}\mathcal{D}^w(x,y)\alpha^{y=1}\left[f(x) \neq y\right] \tag{6.2}$$

$$= \alpha\sum_{x\in\mathcal{X}}\left(\mathcal{D}^w(x,+1)\left[f(x) \neq +1\right] + \mathcal{D}^w(x,-1)\frac{1}{\alpha}\left[f(x) \neq -1\right]\right) \tag{6.3}$$

$$= \alpha\sum_{x\in\mathcal{X}}\left(\mathcal{D}^b(x,+1)\left[f(x) \neq +1\right] + \mathcal{D}^b(x,-1)\left[f(x) \neq -1\right]\right) \tag{6.4}$$

$$= \alpha\mathbb{E}_{(x,y)\sim\mathcal{D}^b}\left[f(x) \neq y\right] \tag{6.5}$$

$$= \alpha\epsilon^b \tag{6.6}$$

STRATEGIES FOR INDUCING A NEW BINARY DISTRIBUTION

- Undersample the negative class
- Oversample the positive class

STRATEGIES FOR INDUCING A NEW BINARY DISTRIBUTION

- Undersample the negative class
  - More computationally efficient
- Oversample the positive class
  - Base binary classifier might do better with more training examples
  - Efficient implementations incorporate weight in algorithm, instead of explicitly duplicating data

Algorithm 1 DecisionTreeTrain(*data*, *remaining features*)

```
1:  guess ← most frequent answer in data          // default answer for this data
2:  if the labels in data are unambiguous then
3:      return Leaf(guess)                        // base case: no need to split further
4:  else if remaining features is empty then
5:      return Leaf(guess)                        // base case: cannot split further
6:  else                                          // we need to query more features
7:      for all f ∈ remaining features do
8:          NO ← the subset of data on which f=no
9:          YES ← the subset of data on which f=yes
10:         score[f] ← # of majority vote answers in NO
11:                  + # of majority vote answers in YES
                                  // the accuracy we would get if we only queried on f
12:     end for
13:     f ← the feature with maximal score(f)
14:     NO ← the subset of data on which f=no
15:     YES ← the subset of data on which f=yes
16:     left ← DecisionTreeTrain(NO, remaining features \ {f})
17:     right ← DecisionTreeTrain(YES, remaining features \ {f})
18:     return Node(f, left, right)
19: end if
```

REDUCTIONS

- Idea is to re-use simple and efficient algorithms for binary classification to perform more complex tasks
- Works great in practice

LEARNING WITH IMBALANCED DATA IS AN EXAMPLE OF REDUCTION

TASK: α-WEIGHTED BINARY CLASSIFICATION

Given:

1. An input space $\mathcal{X}$
2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$
3. A training set D sampled from $\mathcal{D}$

Compute: A function f minimizing: $\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\alpha^{y=1}\left[f(\boldsymbol{x}) \neq y\right]\right]$

Subsampling Optimality Theorem:

If the binary classifier achieves a binary error rate of ε, then the error rate of the α-weighted classifier is α ε

MULTICLASS CLASSIFICATION

- Real world problems often have multiple classes (text, speech, image, biological sequences…)
- How can we perform multiclass classification?
  - Straightforward with decision trees or KNN
  - Can we use the perceptron algorithm?

REDUCTIONS FOR MULTICLASS CLASSIFICATION

TASK: MULTICLASS CLASSIFICATION

Given:

1. An input space $\mathcal{X}$ and number of classes K
2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times [K]$
3. A training set D sampled from $\mathcal{D}$

Compute: A function f minimizing: $\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[f(\boldsymbol{x}) \neq y\right]$

TASK: BINARY CLASSIFICATION

Given:

1. An input space $\mathcal{X}$
2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$
3. A training set D sampled from $\mathcal{D}$

Compute: A function f minimizing: $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[f(\boldsymbol{x}) \neq y\right]$

HOW MANY CLASSES CAN WE HANDLE?

- In most tasks, number of classes K < 100
- For much larger K
  - we need to frame the problem differently
  - e.g, machine translation or automatic speech recognition

WHAT YOU SHOULD KNOW

- How can we take the standard binary classifier and adapt it to handle problems with
  - Imbalanced data distributions
  - Multiclass classification problems
- Algorithms & guarantees on error rate
- Fundamental ML concept
  - Reduction

REDUCTION 1: OVA

- "One versus all" (aka "one versus rest")
  - Train K-many binary classifiers
  - Classifier k predicts whether an example belong to class k or not
  - At test time
    - If only one classifier predicts positive, predict that class
    - Break ties randomly

Algorithm 13 OneVersusAllTrain($\mathbf{D}^{multiclass}$, BinaryTrain)

1: **for** $i = 1$ **to** K **do**
2: $\quad \mathbf{D}^{bin} \leftarrow$ relabel $\mathbf{D}^{multiclass}$ so class i is positive and $\neg i$ is negative
3: $\quad f_i \leftarrow$ BinaryTrain($\mathbf{D}^{bin}$)
4: **end for**
5: **return** $f_1, \ldots, f_K$

Algorithm 14 OneVersusAllTest($f_1, \ldots, f_K, \hat{\boldsymbol{x}}$)

1: $score \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize K-many scores to zero
2: **for** $i = 1$ **to** K **do**
3: $\quad y \leftarrow f_i(\hat{\boldsymbol{x}})$
4: $\quad score_i \leftarrow score_i + y$
5: **end for**
6: **return** $\text{argmax}_k \; score_k$

TIME COMPLEXITY

- Suppose you have N training examples, in K classes. How long does it take to train an OVA classifier
  - If the base binary classifier takes O(N) time to learn?
  - If the base binary classifier takes O(N^2) time to learn?

ERROR BOUND

- **Theorem:** Suppose that the average error of the K binary classifiers is ε, then the error rate of the OVA multiclass classifier is at most $(K-1)\,\varepsilon$
- To prove this: how do different binary classifier errors affect the probability of a multiclass error?

ERROR BOUND PROOF

- If we have a **false negative** on one of the binary classifiers (assuming all other classifiers correctly output negative)
- What is the probability that we will make an incorrect multiclass prediction?

$$(K - 1) / K$$

Efficiency: $[(K - 1) / K] / 1 = (K - 1) / K$

(maximum ratio of the probability of a multiclass error to the number of binary errors)

ERROR BOUND PROOF

- If we have k **false positives** with the binary classifiers
- What is the probability that we will make an incorrect multiclass prediction?
  - If there is also a false negative: 1
    - Efficiency =1 / k + 1
  - Otherwise k / (k + 1)
    - Efficiency = [k / (k + 1)] / k = 1 / (k + 1)

ERROR BOUND PROOF

- What is the worst case scenario?
  - False negative case: efficiency is (K-1)/K
    - Larger than false positive efficiencies
  - There are K-many opportunities to get false negative, **overall error bound is (K-1) ε**

REDUCTION 2: AVA

- All versus all (aka all pairs)
- How many binary classifiers does this require?

Algorithm 15 AllVersusAllTrain($\mathbf{D}^{multiclass}$, BinaryTrain)

$f_{ij} \leftarrow \emptyset, \forall 1 \leq i < j \leq K$
for $i = 1$ **to** K-1 **do**
  $\mathbf{D}^{pos} \leftarrow$ all $\boldsymbol{x} \in \mathbf{D}^{multiclass}$ labeled i
  for $j = i+1$ **to** K **do**
    $\mathbf{D}^{neg} \leftarrow$ all $\boldsymbol{x} \in \mathbf{D}^{multiclass}$ labeled j
    $\mathbf{D}^{bin} \leftarrow \{(\boldsymbol{x}, +1) : \boldsymbol{x} \in \mathbf{D}^{\text{pos}}\} \cup \{(\boldsymbol{x}, -1) : \boldsymbol{x} \in \mathbf{D}^{\text{neg}}\}$
    $f_{ij} \leftarrow$ BinaryTrain($\mathbf{D}^{bin}$)
  end for
end for
return all f_{ij}s

Algorithm 16 AllVersusAllTest(all f_{ij}, $\hat{\boldsymbol{x}}$)

$score \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize K-many scores to zero
for $i = 1$ **to** K-1 **do**
  for $j = i+1$ **to** K **do**
    $y \leftarrow f_{ij}(\hat{\boldsymbol{x}})$
    $score_i \leftarrow score_i + y$
    $score_j \leftarrow score_j - y$
  end for
end for
return $\text{argmax}_k \; score_k$

TIME COMPLEXITY

- Suppose you have N training examples, in K classes. How long does it take to train an AVA classifier
  - if the base binary classifier takes O(N) time to learn?
  - if the base binary classifier takes O(N^2) time to learn?

ERROR BOUND

- **Theorem:** Suppose that the average error of the K binary classifiers is ε, then the error rate of the AVA multiclass classifier is at most $2(K-1)\,\varepsilon$

- Question: Does this mean that AVA is always worse than OVA?

SUMMARY

- Given an arbitrary method for binary classification, how can we learn to make multiclass predictions?
  - OVA, AVA
- Fundamental ML concept: reductions

ANNOUNCEMENTS

- Grades for Homework 1 have been published
- Updated hw2.zip on the schedule
  - Test cases match the visible test cases on the server (changes in tests_simple & run_tests_simple.py)
  - Directions on what to submit (changes in hw1.ipynb)
    - Upload written part separately as a PDF
  - Written assignments are marked more clearly (hw1.ipynb & data/kitten.jpeg)

ACKNOWLEDGEMENTS

- These slides use materials by Marine Carpuat, Brian Ziebart