Supervised Clustering
Learning a Similarity Measure from Example Clusterings

Thomas Finley

CS630 November 1, 2004

Supervised Clustering Talk Outline

- Clustering overview
- Supervised clustering motivation
- Learning to cluster
- Learning to cluster with SVM^{struct}
- Application to real problems

Supervised Clustering Talk Outline

- Clustering overview
- Supervised clustering motivation
- Learning to cluster
- Learning to cluster with SVM^{struct}
- Application to real problems

Simple Clustering

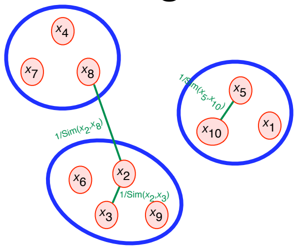
- We have a set of m items.

$$\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$$

- We have a similarity measure between pairs of items.

$$Sim(x_i, x_j)$$

- We want to produce a partitioning of the item set.



$$\mathbf{y} = \{y_1, y_2, \ldots, y_c\} \qquad \forall y_i \in \mathbf{y} : y_i \subseteq \mathbf{x}$$
$$\bigcup_{y_i \in \mathbf{y}} y_i = \mathbf{x} \qquad \forall y_i, y_j \in \mathbf{y} : y_i \cap y_j = \emptyset$$

Simple Clustering

- Clustering algorithms find the partition $\mathbf{y}$ that maximizes some objective function with respect to this similarity measure.
- For example, an objective function could find the clustering that maximizes the sum of the similarity of items in the same cluster.

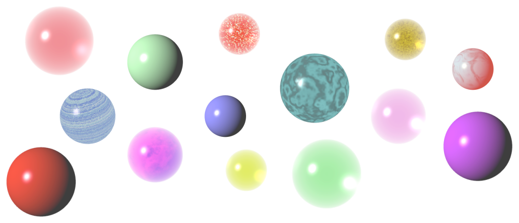
$$\frac{1}{n^2}\sum_{y_k\in\mathbf{y}}\sum_{x_i,x_j\in y_k} Sim(x_i,x_j)$$

- Other clustering objective functions exist, but this is the example for the talk.

Supervised Clustering Talk Outline

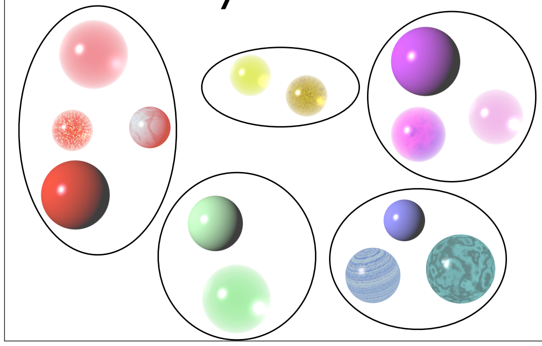
- Clustering overview
- Supervised clustering motivation
- Learning to cluster
- Learning to cluster with SVM^{struct}
- Application to real problems

Clustering Marbles

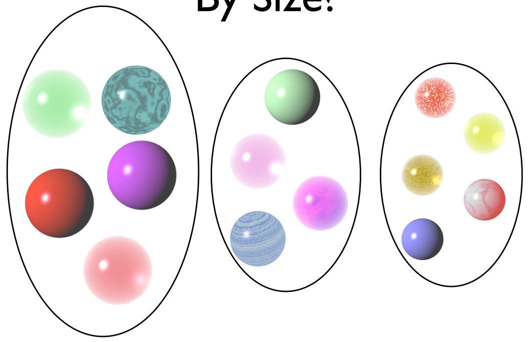


By what criteria do we consider two marbles "similar"?

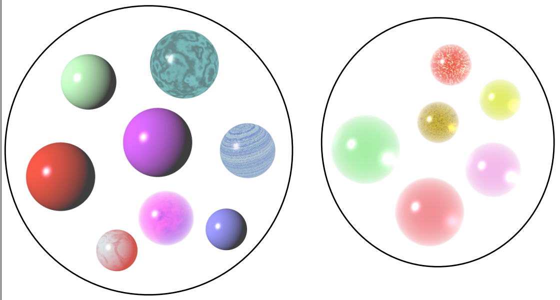
By Color?



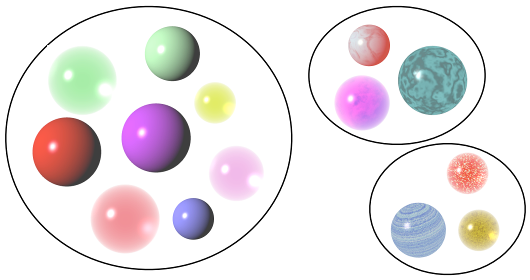
By Size?



By Transparency?



The "Busy-ness" of the Surface Pattern?



It's Unclear!

- There are almost always multiple reasonable ways of partitioning a set of items.
- Without further information, there's no unambiguously correct partitioning.
- What do I mean by "further information?"

Manual Adjustment

- One could manually adjust the clustering algorithm to produces desirable clusterings.
- However, some form of automatic adjustment might produce better results.

Supervised Clustering

- One could provide a series of sets of items, and the complete corresponding clusterings.
- The algorithm adjusts itself so that future sets of items you provide are partitioned in the same fashion as training instances.
- This is the setting we're interested in.

Supervised Clustering

- This is useful for settings where you have a series of small separately clustered sets.
- Grouping a day's worth of news stories according to which have the same topic (Google News).
- Grouping noun-phrases in documents by which talk about the same entity (NP coreference).

Supervised Clustering Talk Outline

- Clustering overview
- Supervised clustering motivation
- Learning to cluster
- Learning to cluster with SVM^{struct}
- Application to real problems

How Do We Learn?

- We suppose we have a sequence of *n* training examples.

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$$

- Each **x** corresponds to a set of items, and each **y** is a partitioning over that set.
- From this training data we learn a similarity measure for pairs of items.

Why Similarity?

- Suitability of a clustering depends more on the similarity measure than the clustering method.
- If you want marbles clustered by size, a similarity measure that stresses color similarity won't produce suitable clusterings.
- On the other hand, a measure that stresses size similarity will tend to produce good results even on a mediocre clusterer.

Similarity

- For each item *i* and *j*, we have a pairwise feature vector ϕ_{ij}.
- For marbles, features may include hue angle, diameter, and transparency differences, as well as a single "bias" feature that is constant for all pairs of items.
- Similarity between items i and j is an inner product between a learned vector of weights, and the vector of pairwise features.

$$Sim(x_i, x_j) = \langle \mathbf{w}, \phi_{ij} \rangle$$

Similarity

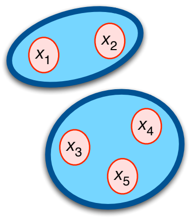
- Change **w** to change the similarity measure.
- For example, if an element of ϕ_{ij} is important for determining the similarity between items *i* and *j*, the corresponding element in **w** should probably have a higher magnitude.

Naïve Way to Learn

- Can a binary classifier to learn to cluster?
- For a training example ($\mathbf{x}_i$,$\mathbf{y}_i$), and each pair x_j,x_k in $\mathbf{x}_i$, if they are in the same cluster in $\mathbf{y}_i$, ϕ_{jk} is a positive training example "in the same cluster."
- Else, ϕ_{jk} is a negative training example.
- Train, and let the output of the learned function be the pairwise similarity measure.

Simple Example

- Set $\mathbf{x}$ with partitioning $\mathbf{y}$.
- Positive examples: ϕ_{12}, ϕ_{34}, ϕ_{35}, ϕ_{45}.
- Negative examples: ϕ_{13}, ϕ_{14}, ϕ_{15}, ϕ_{23}, ϕ_{24}, ϕ_{25}.
- Linear SVM trained on these will learn a weight vector $\mathbf{w}$.



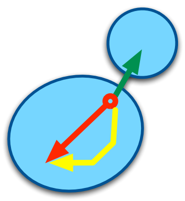
$$Sim(x_i,x_j) = \langle \mathbf{w}, \phi_{ij} \rangle$$

What About Differing Performance Measures?

- In most clustering tasks, the vast number of pairwise relationships are negative.
- So, a learned hypothesis would understate whether a pair is in the same cluster.
- This is bad for tasks where the canonical performance measure has a bias towards putting everything in one cluster.
- E.g., MITRE measure for NP coreference.

What About Cluster Interactions?

- A pairwise classifier may try too hard to make the red similarity high, getting the more critical green similarity wrong.
- A more aware learner could be comfortable keeping the red similarity low. The nodes would be in the same cluster due to an indirect interaction with nodes on the yellow path.



Avoiding Indirection

- As a matter of principle, which is more attractive?
- To learn a hypothesis optimized for pairwise decisions and hope it's good for clustering…
- …or to learn a hypothesis optimized for clustering?

Supervised Clustering Talk Outline

- Clustering overview
- Supervised clustering motivation
- Learning to cluster
- Learning to cluster with SVM*struct*
- Application to real problems

SVM*struct* Overview

- Often SVMs are used to learn functions that map to a binary output, as in binary classification.
- SVM*struct* is an adaptation of SVM*light* for learning functions with a more complex output space.

Quadratic Program Formulation

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } \forall i : \xi_i \geq 0$$
$$\forall i, \forall \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\} : \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i)\rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y})\rangle \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i$$

- Pretty much a vanilla SVM except for the last type of constraint, a generalization for tasks with complex output.
- Requires a function relating the input and the output Ψ and a loss function Δ.

For those familiar with this: I use the linear margin scaling variant of the QP throughout this presentation.

SVM*struct* and Complex Output Spaces

- Map a sequence of words to a parse tree.
- Map a query to a ranking of documents.
- Can we map a set of items to a partitioning over that set?
- Maybe. What sort of restrictions does SVM*struct* have for the types of problems it handles?

Problem Requirements

- We need a problem where we map an input **x** to some output **y**, according to a function *f* parameterized by some weight vector **w**:

$$f(\mathbf{x};\mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle$$

- $\Psi(\mathbf{x},\mathbf{y})$ characterizes the relationship between an input **x** and an output **y**.
- Put simply, SVM*struct* finds the parameter vector **w** such that for the training examples $(\mathbf{x}_i,\mathbf{y}_i)$, $f(\mathbf{x}_i;\mathbf{w})$ is $\mathbf{y}_i$ (or similar).

Ψ for Clustering

- We want a partitioning **y** over a set **x** that maximizes a clustering objective function.

$$\operatorname*{argmax}_{\mathbf{y}} \frac{1}{n^2} \sum_{y_k\in\mathbf{y}} \left(\sum_{x_i,x_j\in y_k} Sim(x_i,x_j) \right)$$

- The similarity between two items in **x** is an inner product between **w** and pairwise features.

$$Sim(x_i,x_j) = \langle \mathbf{w}, \phi_{ij} \rangle$$

- Does this jibe with the type of concepts SVM$^{\text{struct}}$ can learn?

$$f(\mathbf{x};\mathbf{w}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle$$

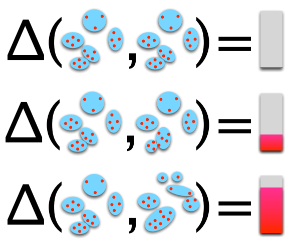
Ψ for Clustering

$$\begin{aligned}
\operatorname*{argmax}_{\mathbf{y}} \frac{1}{n^2} \sum_{y_k\in\mathbf{y}} \left(\sum_{x_i,x_j\in y_k} Sim(x_i,x_j) \right) &= \operatorname*{argmax}_{\mathbf{y}} \frac{1}{n^2} \sum_{y_k\in\mathbf{y}} \left(\sum_{x_i,x_j\in y_k} \langle \mathbf{w}, \phi_{ij} \rangle \right) \\
&= \operatorname*{argmax}_{\mathbf{y}} \left\langle \mathbf{w}, \frac{1}{n^2} \sum_{y_k\in\mathbf{y}} \left(\sum_{x_i,x_j\in y_k} \phi_{ij} \right) \right\rangle \\
&= \operatorname*{argmax}_{\mathbf{y}} \langle \mathbf{w}, \Psi(\mathbf{x},\mathbf{y})\rangle
\end{aligned}$$

- Clustering fulfills the basic requirement to be a concept learnable within the SVM*struct* framework.
- What else do we need to learn?

Δ for Clustering

- Δ indicates how unrelated two clusterings $\mathbf{y}_i$ and $\mathbf{y}_j$ are.
- If $\mathbf{y}_i$=$\mathbf{y}_j$ then Δ($\mathbf{y}_i$, $\mathbf{y}_j$)=0.
- SVM$^{\text{struct}}$ will prefer $\mathbf{w}$ for which $f(\mathbf{x}_i;\mathbf{w})$ produces a clustering with the smallest possible loss relative to the training example $\mathbf{y}_i$.



Δ for Clustering

- What are possible Δ($\mathbf{y}_i$,$\mathbf{y}$)?
- Pairwise loss. Over all item pairs, what ratio to $\mathbf{y}_i$ and $\mathbf{y}$ disagree about whether they're in the same partition?
- Mutual clustering purity. Find the purity of $\mathbf{y}_i$ versus $\mathbf{y}$, and of $\mathbf{y}$ versus $\mathbf{y}_i$, and take the harmonic mean.
- MITRE measure. Used for NP coreference.

Quadratic Program Formulation Revisited

$$\min_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } \forall i : \xi_i \geq 0$$
$$\forall i, \forall \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\} : \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y}_i)\rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y})\rangle \geq \Delta(\mathbf{y}_i,\mathbf{y}) - \xi_i$$

- The last constraint is where the magic is.
- Now that we know what Ψ and Δ are, how can we interpret this constraint?

Last Constraint?

$$\forall i, \forall \mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}_i\} : \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y}_i)\rangle \geq \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y})\rangle + \Delta(\mathbf{y}_i,\mathbf{y}) - \xi_i$$

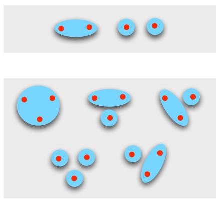
- For all training examples, and for all possible wrong clusterings, keep the value of the objective function for the correct clustering greater than the value of the objective function for every wrong clustering, and make sure they differ by at least the loss between the right and wrong clustering. We allow some flexibility if appropriate.

Great!

- Ignoring the ξ slack variables, we can pick a **w** such that the clusterer will always pick the correct clustering!
- If it is necessary to use the slack variables, we at least encourage clusterings with low loss relative to the training cluster.
- All we need to do is introduce a constraint for every single wrong partitioning for every training example.

OK…How Many is That?

- A 2 item training example has 2 ways to partition, so 1 constraint.
- A set with 3 items has 5 ways to partition, so 4 constraints.
- How many constraints for a 1000 item set?



How Many Constraints?

- 29,899,013,356,824,084,214,804,223,538,97
6,464,839,473,928,098,212,305,047,832,737,
888,945,413,625,123,259,596,641,165,872,5
40,391,578,300,639,147,082,986,964,028,02
1,802,248,993,382,881,013,411,276,574,829,
121,155,811,755,170,830,666,039,838,837,2
73,971,971,676,782,389,800,810,361,809,31
9,250,755,399,325,279,656,765,435,255,999,
301,529,770,267,107,281,619,733,800,…

How Many Constraints? Page 2

- 281,695,881,540,007,577,899,106,878,679,4
51,165,492,535,930,459,233,713,316,342,55
1,545,242,815,802,367,257,284,852,612,201,
081,016,386,308,535,990,145,447,341,800,4
55,472,334,713,864,080,523,978,960,296,36
5,736,999,295,932,080,550,928,561,633,025,
800,627,524,911,700,149,562,106,895,897,7
25,047,744,775,812,241,800,937,310,491,79
7,818,107,578,233,924,187,312,824,632,…

How Many Constraints? Page 3

- 629,095,993,832,334,781,713,007,323,483,6
88,294,825,326,897,450,386,817,327,410,53
2,925,074,613,888,321,264,138,083,842,196,
202,242,956,001,314,953,449,497,244,271,8
43,922,741,908,252,107,652,201,346,933,88
9,741,070,435,350,690,242,062,001,522,697,
855,278,356,012,055,718,392,851,567,813,3
97,125,419,144,780,476,479,197,990,921,60
2,015,873,703,820,769,182,603,836,788,…

How Many Constraints? Page 4

- 465,785,093,563,686,025,690,269,802,153,8
02,436,873,530,877,006,737,154,523,895,27
3,029,510,238,745,997,356,292,232,631,282,
773,748,762,989,386,003,970,214,423,843,9
47,094,021,177,989,737,557,020,369,751,56
1,595,003,372,955,621,411,858,485,959,813,
344,799,967,960,196,238,368,337,022,346,9
46,771,703,060,269,288,691,694,028,444,79
1,203,978,533,454,759,410,587,065,022,…

How Many Constraints? Page 5

- 546,491,518,871,238,421,560,825,907,135,8
85,619,221,776,405,898,771,057,270,555,58
1,449,229,994,215,739,476,758,785,884,545,
723,062,263,992,367,750,091,319,644,861,5
47,658,472,282,284,005,892,044,371,587,56
0,711,880,627,741,139,497,818,835,632,120,
761,570,174,928,529,697,397,267,899,554,4
07,350,161,283,097,123,211,048,049,269,72
7,655,279,783,900,702,416,095,132,827,…

How Many Constraints? Page 6

- 766,428,865,017,653,366,696,304,131,436,6
90,232,979,453,876,337,599,721,772,897,04
9,270,230,544,262,611,264,917,393,374,756,
384,152,784,943,607,952,408,782,612,639,2
20,380,791,445,272,655,004,475,989,064,27
6,373,713,608,901,650,681,165,467,490,310,
898,804,916,827,069,427,310,961,109,285,0
35,545,084,791,339,423,266,482,359,955,66
3,377,201,515,204,340,817,580,915,468,…

How Many Constraints? Page 7

- 489,969,181,643,341,007,197,836,481,461,0
51,798,995,640,789,292,580,146,918,580,70
3,759,556,634,019,451,731,530,034,209,189,
203,377,522,668,309,771,129,566,108,101,6
17,727,442,045,637,098,112,678,864,654,30
9,987,785,463,307,376,544,339,506,878,267,
267,349,348,171,320,834,971,956,806,668,3
04,099,159,992,067,385,998,690,820,326,90
2,473,886,782,781,499,414,773,178.

That's a lot of constraints!

- That's about 2.99×10^{1927} constraints necessary for a 1000 item training example.
- There are only an estimated 10^{78} to 10^{81} atoms in the universe. So, we need to work on subatomic media to store them.
- At 500 cycles per constraint, that's 4.3×10^{1894} ages-of-the-universe to generate them on my machine.
- No problem!

Big Problem

- SVM^{struct} uses only a select few of these possible constraints to form an approximation to the full quadratic problem.
- With these select constraints, we find a **w** where no constraint in the full program is violated by more than an arbitrary ϵ.
- How do we pick this "select few?"

Algorithm in Detail

- Now we will finally go over the algorithm!

Algorithm in Detail

- The algorithm gets input of *n* training examples.
- Regularization parameter *C* to control how much slack it will tolerate.
- Tolerance ϵ to indicate how far from the true optimum it may stray.
- With no meaningful constraints yet, **w** is zero.

```
Input: $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1, \ldots, n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i \rangle$
    compute $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y} \in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- Recall it only introduces constraints for *some* of the wrong outputs.
- Every training example has a set indicating outputs for which it already has constraints.
- At the start of the algorithm this set is empty.

```
Input: $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1, \ldots, n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i \rangle$
    compute $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y} \in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- We repeatedly iterate over all training examples.

```
Input: $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1, \ldots, n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i \rangle$
    compute $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y} \in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- A function *H* is provided that takes an output **y**.
- *H* returns slack ξ that would be required if we introduced a constraint for this **y** under the current learned **w**.

```
Input: $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1, \ldots, n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i \rangle$
    compute $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y} \in \mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y} \in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- We find the value $\hat{\mathbf{y}}$ associated with the most violated constraint.
- This term is a constant.
- This is our clustering objective function.
- So, we must produce a variant of the clustering function that optimizes loss plus objective function to find this argmax.

```
Input: $(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_n,\mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1,\ldots,n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i,\mathbf{y}) +$ 🟦 $-$ 🟩
    compute $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y}\in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- Find the slack associated with this training example over all wrong outputs for which we already have constraints.

```
Input: $(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_n,\mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1,\ldots,n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i,\mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y})\rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y}_i)\rangle$
    compute $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y}\in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- See if the output for the currently most violated example requires more slack than is already being provided, within a tolerance.
- If it doesn't, this constraint can't change the solution, so we don't need to consider it further.

```
Input: $(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_n,\mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1,\ldots,n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i,\mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y})\rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y}_i)\rangle$
    compute $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y}\in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- If it does, we should add a constraint to the Q.P.
- We add the output to the outputs for which we have constraints.

```
Input: $(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_n,\mathbf{y}_n)$, $C$, $\epsilon$
$S_i \leftarrow \emptyset$ for all $i = 1,\ldots,n$
repeat
  for i=1, ..., n do
    set up a cost function
    $H(\mathbf{y}) = \Delta(\mathbf{y}_i,\mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y})\rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i,\mathbf{y}_i)\rangle$
    compute $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} H(\mathbf{y})$
    compute $\xi_i = \max\{0, \max_{\mathbf{y}\in S_i} H(\mathbf{y})\}$
    if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
      $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
      $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
    end if
  end for
until no $S_i$ has changed during iteration
```

Algorithm in Detail

- The constraint introduced will change the solution.
- So, we find the new **w** for Q.P. this new constraint added.

```
1:  Input: $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, $C$, $\epsilon$
2:  $S_i \leftarrow \emptyset$ for all $i = 1, \ldots, n$
3:  repeat
4:    for i=1, ..., n do
5:      set up a cost function
        $H(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i) \rangle$
6:      compute $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} H(\mathbf{y})$
7:      compute $\xi_i = \max\{0, \max_{\mathbf{y} \in S_i} H(\mathbf{y})\}$
8:      if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
9:        $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
10:       $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
11:     end if
12:   end for
13: until no $S_i$ has changed during iteration
```

Algorithm in Detail

- When the iteration over all training examples produces no more new constraints, there's no point in continuing.
- At this point we stop and return the learned **w**.

```
1:  Input: $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, $C$, $\epsilon$
2:  $S_i \leftarrow \emptyset$ for all $i = 1, \ldots, n$
3:  repeat
4:    for i=1, ..., n do
5:      set up a cost function
        $H(\mathbf{y}) = \Delta(\mathbf{y}_i, \mathbf{y}) + \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}) \rangle - \langle \mathbf{w}, \Psi(\mathbf{x}_i, \mathbf{y}_i) \rangle$
6:      compute $\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} H(\mathbf{y})$
7:      compute $\xi_i = \max\{0, \max_{\mathbf{y} \in S_i} H(\mathbf{y})\}$
8:      if $H(\hat{\mathbf{y}}) > \xi_i + \epsilon$ then
9:        $S_i \leftarrow S_i \cup \{\hat{\mathbf{y}}\}$
10:       $\mathbf{w} \leftarrow$ solution to Q.P. with constraints for $\bigcup_i S_i$
11:     end if
12:   end for
13: until no $S_i$ has changed during iteration
```

Supervised Clustering Talk Outline

- Clustering overview
- Supervised clustering motivation
- Learning to cluster
- Learning to cluster with SVM^{struct}
- Application to real problems

Applications

- We discuss two problems.
- Clustering a day's worth of news stories according to which talk about the same topic.
- Clustering the noun-phrases in a document according to which refer to the same entity.

News Story Clustering

- Unfortunately there aren't many data sets for supervised clustering since it isn't really a problem that has been looked at very much so far.
- We built our own dataset for news stories with the help of Google News.

Trawling Google News

- Every day for thirty days we selected at most 15 news stories from at most 70 related groups -- usually ~900 stories/day.
- The ~900 stories were the items to cluster, and the 70 related groups formed the clusters.

Feature Extraction

- We extract five "regions" from each article:

1. the web page title,
2. the headline (by one heuristic),
3. the headline (by a second heuristic),
4. the article text,
5. and the quoted portions of article text.

Building the pairwise feature vector ϕ

- For each of the five regions, we build three term vectors out of unigram (single words), bigrams, and trigrams. Represents a total of 15 vectors (3 vectors for 5 regions).
- We then compose another 15 vectors that are identical except that the words are Porter stemmed, for a total of 30 vectors.
- Vectors are weighted via TFIDF.

Building the pairwise feature vector ϕ

- For each pair of articles, the pairwise feature vector is of length 31, with the first 30 features representing the cosine similarity between the 30 vectors for each article.
- The last feature is a bias feature that is always 1.

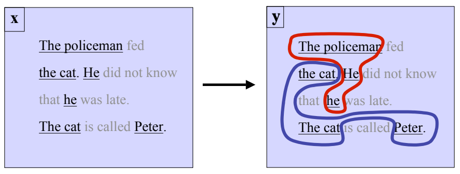
Results in a Nutshell

- The task is very easy to learn.
- The two most highly weighted features are similarity of unigrams for article text, and trigrams for quoted article text.
- Note that trigram is closer to being an equality test for text than unigrams.

Adapting to Processing Errors

- NPR has a paragraph asking for donation. The article text is about two sentences.
- The article extractor considers the donation text more article like than the actual article.
- A few other sites have similar problems.
- Consequence is, while unigram article text similarity is weighted very high, trigram article text similarity is *very* low!

Noun-Phrase Coreference Clustering



NP Coreference

- Data taken from the MUC-6 and -7 task.
- Pairwise features and training data supplied by Vincent Ng and Claire Cardie.
- Used SVM^{light} to learn a binary classifer.
- Used SVM^{struct} to learn a a similarity measure taking clustering performance on the MITRE measure into account.

Results in a Nutshell

- SVM^{struct} beat the pants off SVM^{light}. Why?
- The data is not linearly seperable. A DT can do well, but linear classifiers won't do well.
- The contest became which could better exploit a weakness of the MITRE measure.
- The MITRE measure has a bias towards rewarding overstated pairwise similarities. SVM^{light} understated it. SVM^{struct} optimized for the weakness and won.

This is the end of the talk.