Announcements

Practice Exam Submission

- We opened a practice exam submission (3 questions from the 2019 exam) on gradescope.
- We highly recommend students to try one submission to this practice exam.
- The final exam will have <u>the exact same submission form</u>.

Optimizing Submodular Functions

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu

Recommendations: Diversity

- Redundancy leads to a bad user experience

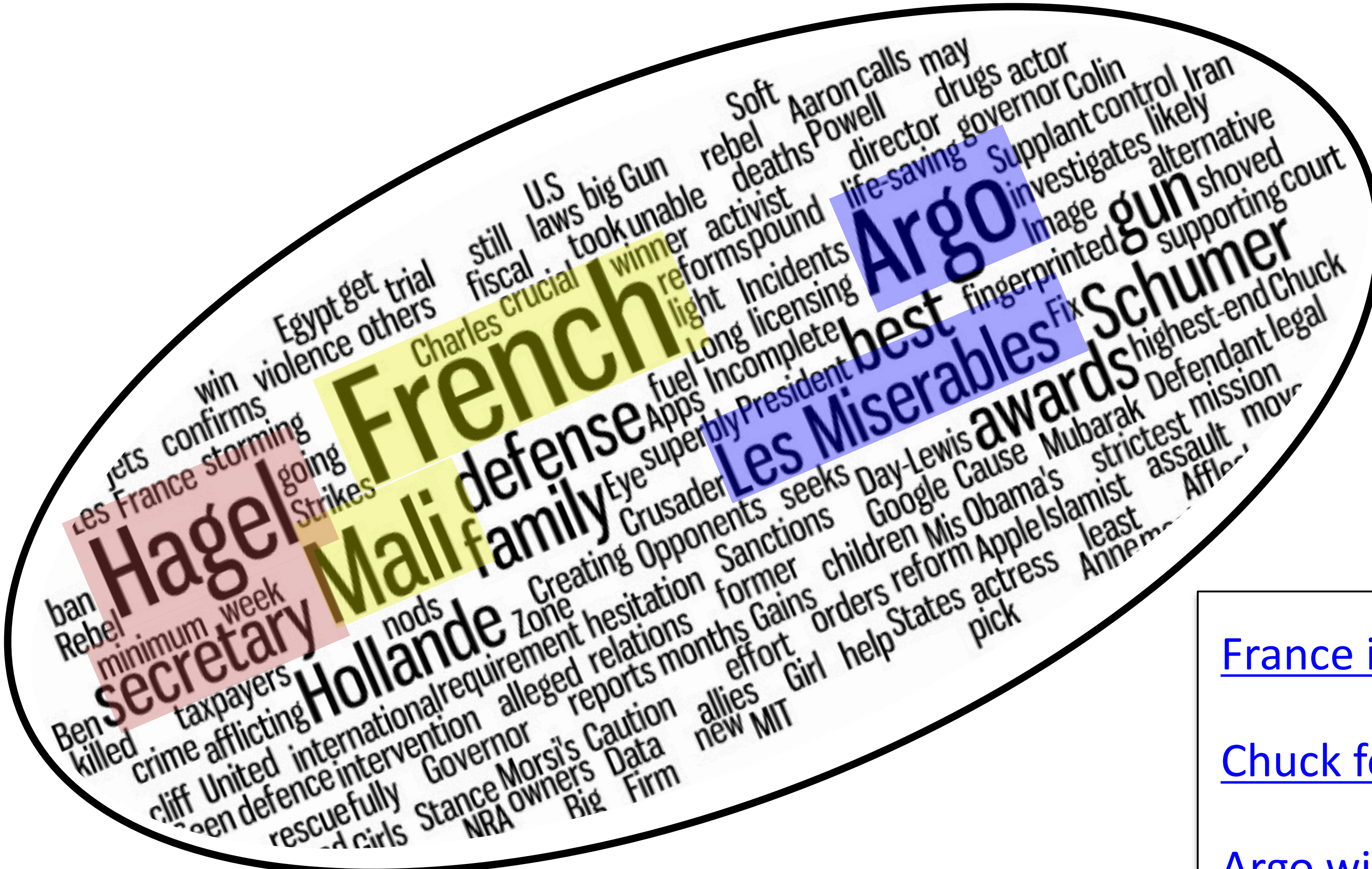
Obama Calls for Broad Action on Guns

Obama unveils 23 executive actions, calls for assault weapons ban

Obama seeks assault weapons ban, background checks on all gun sales

  - Uncertainty around information need => don’t put all eggs in one basket
- How do we optimize for diversity directly?

Covering the day's news



[France intervenes]()

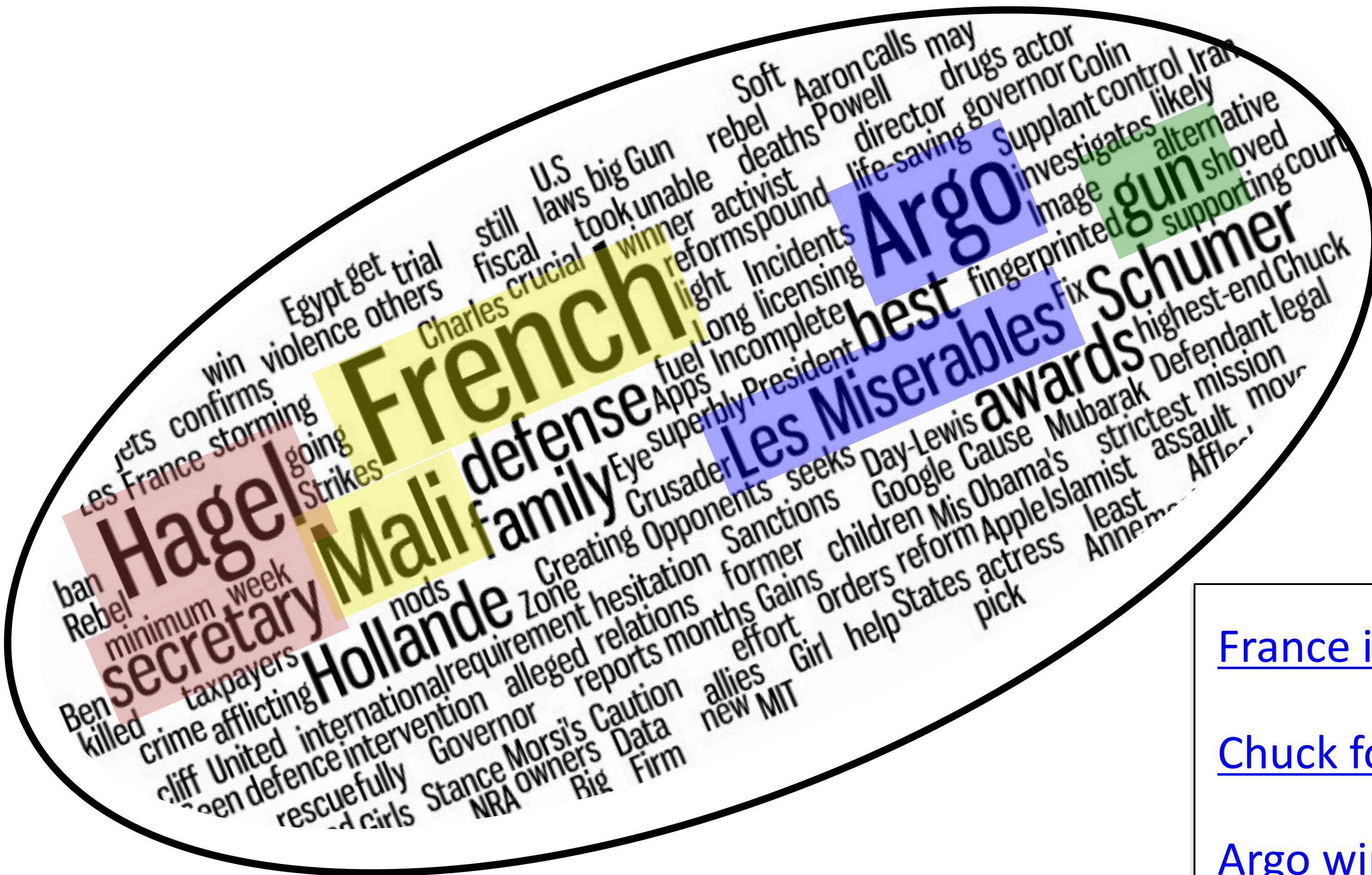
[Chuck for Defense]()

[Argo wins big]()

[Hagel expects fight]()

Monday, January 14

Covering the day's news



[France intervenes]()

[Chuck for Defense]()

[Argo wins big]()

[New gun proposals]()

Monday, January 14

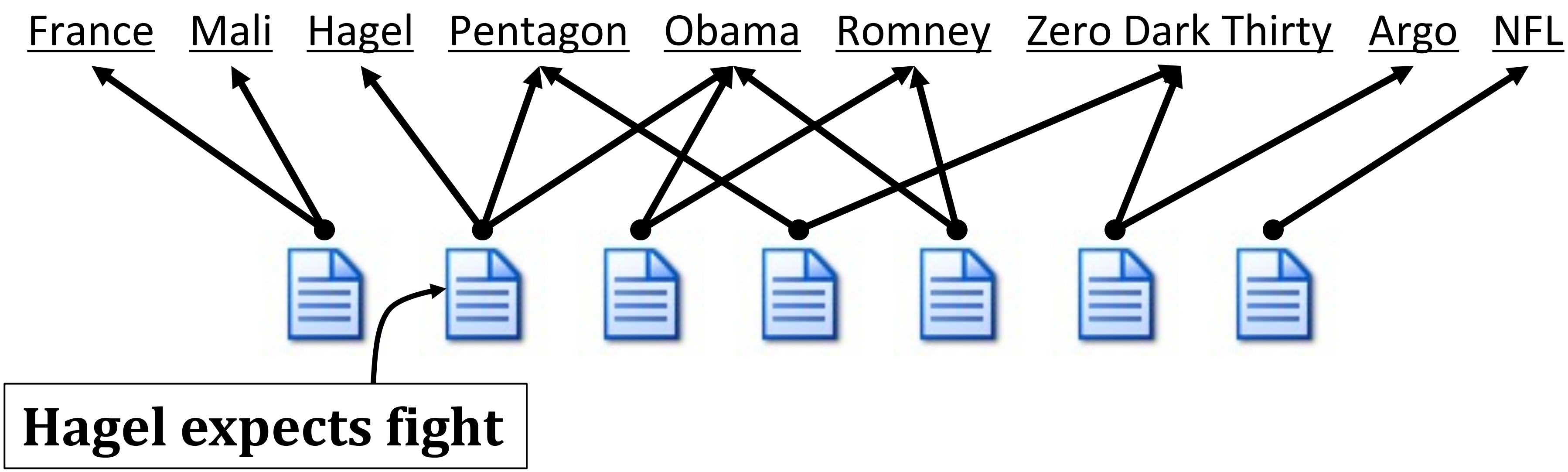
Encode Diversity as Coverage

- **Idea:** Encode diversity as coverage problem
- **Example:** Word cloud of news for a single day
  - Want to select articles so that most words are "covered"

Diversity as Coverage

What is being covered?

- **Q: What is being covered?**
- **A: Concepts** (In our case: Named entities)



- **Q: Who is doing the covering?**
- **A: Documents**

Simple Abstract Model

- **Suppose we are given a set of documents D**
  - Each document **d** covers a set X_d of words/topics/named entities **W**
- **For a set of documents *A* $\subseteq$ *D* we define**

$$F(A) = \left| \bigcup_{i \in A} X_i \right|$$

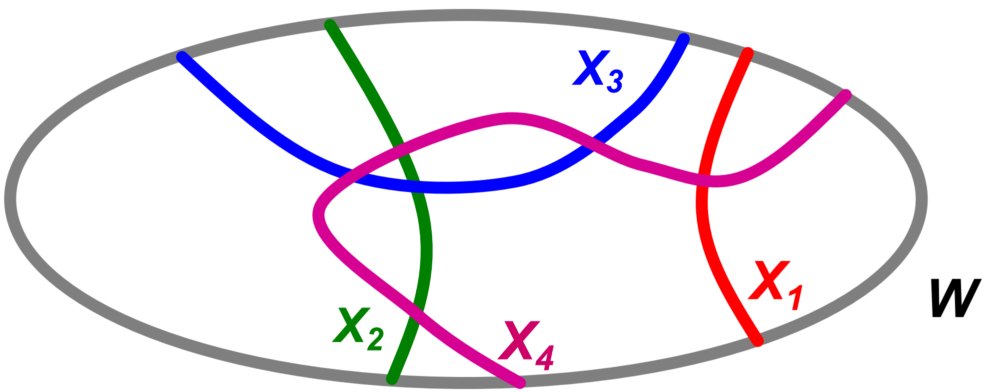
- **Goal: We want to**

$$\max_{|A| \leq k} F(A)$$

- **Note: *F(A)* is a set function: $F(A)$: Sets $\to \mathbb{N}$**

Maximum Coverage Problem

- **Given universe of elements $W = \{w_1, \dots, w_n\}$ and sets $X_1, \dots, X_m \subseteq W$**



- **Goal: Find *k* sets X_i that cover the most of W**
  - **More precisely: Find *k* sets X_i whose size of the union is the largest**
  - **Bad news:** A known NP-complete problem

Simple Greedy Heuristic

Simple Heuristic: Greedy Algorithm:

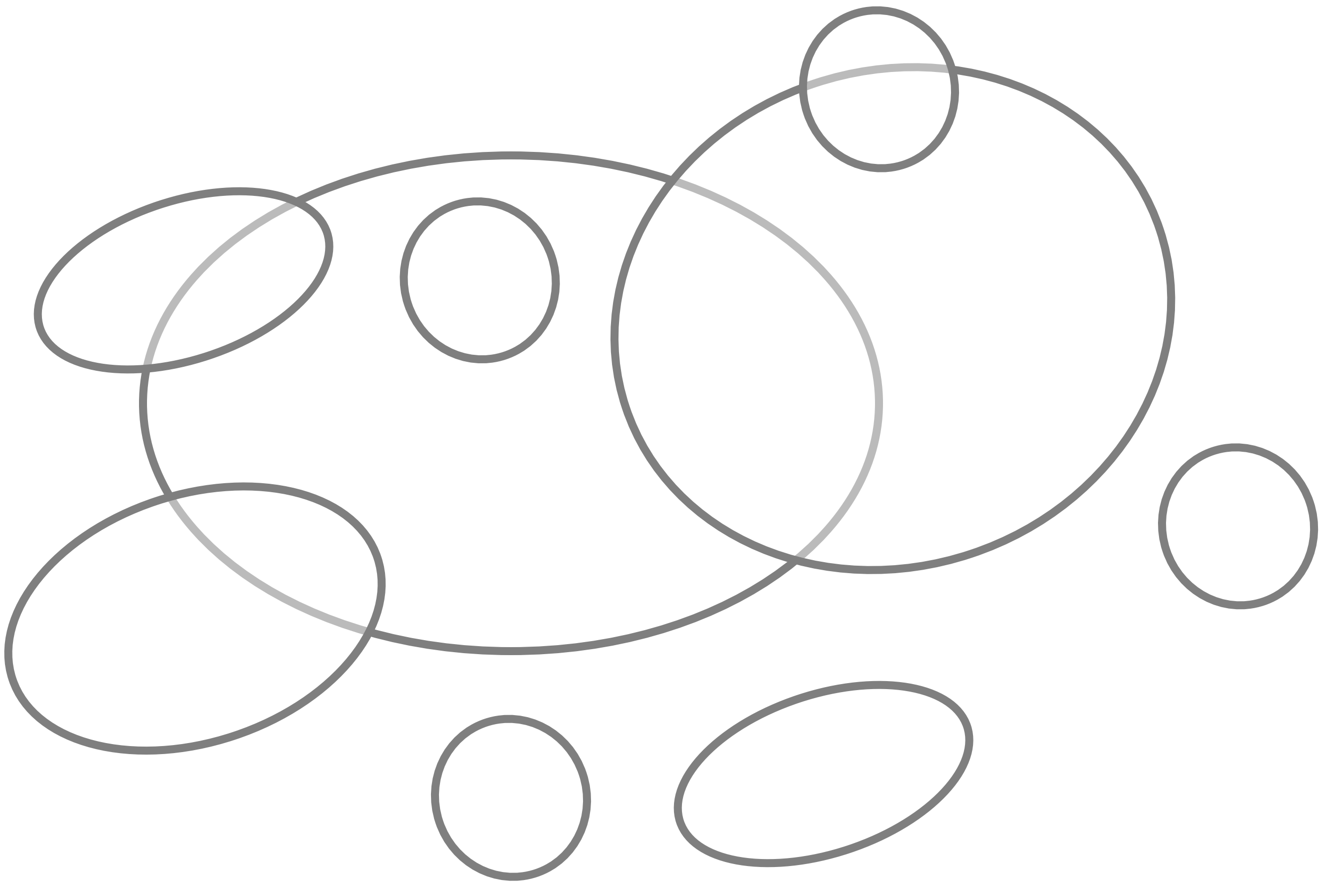
- Start with $A_0 = \{\ \}$
- For $i = 1 \ldots k$
  - Find set d that $\max F(A_{i-1} \cup \{d\})$
  - Let $A_i = A_{i-1} \cup \{d\}$

$$F(A) = \left| \bigcup_{d \in A} X_d \right|$$

- **Example:**
  - Eval. $F(\{d_1\}), \ldots, F(\{d_m\})$, pick best (say d_1)
  - Eval. $F(\{d_1\} \cup \{d_2\}), \ldots, F(\{d_1\} \cup \{d_m\})$, pick best (say d_2)
  - Eval. $F(\{d_1, d_2\} \cup \{d_3\}), \ldots, F(\{d_1, d_2\} \cup \{d_m\})$, pick best
  - And so on…

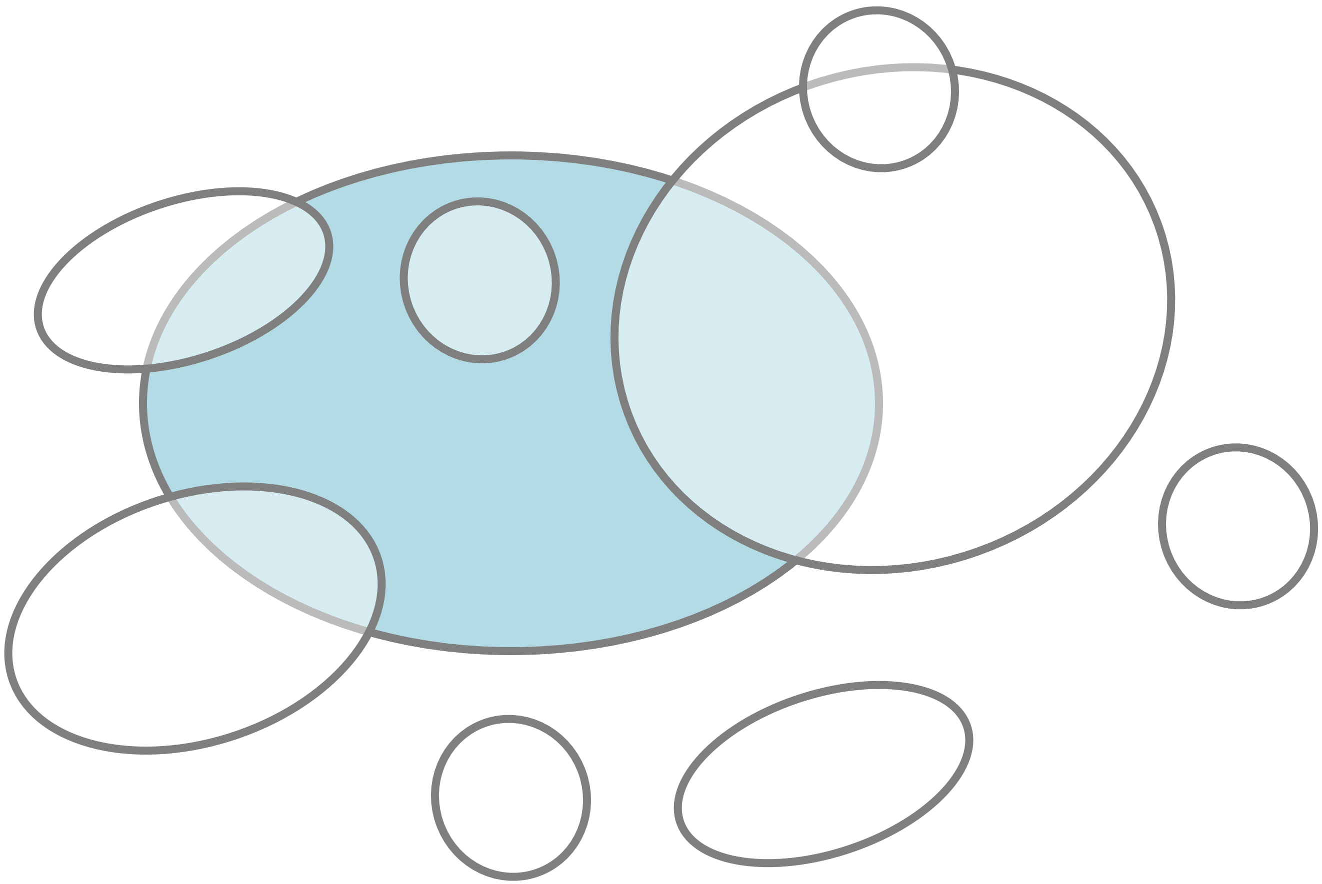
Simple Greedy Heuristic

- **Goal: Maximize the covered area**



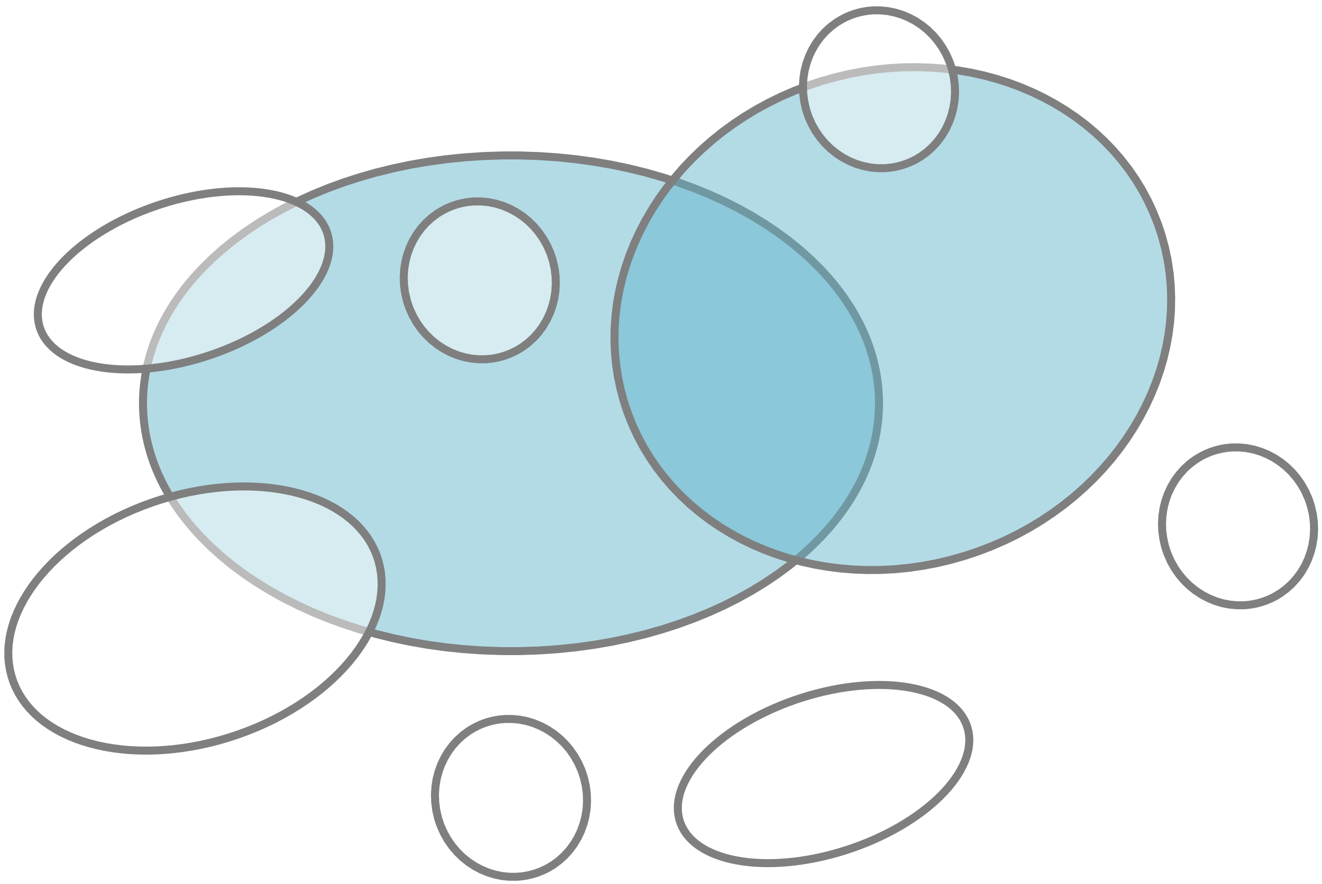
Simple Greedy Heuristic

- **Goal: Maximize the covered area**



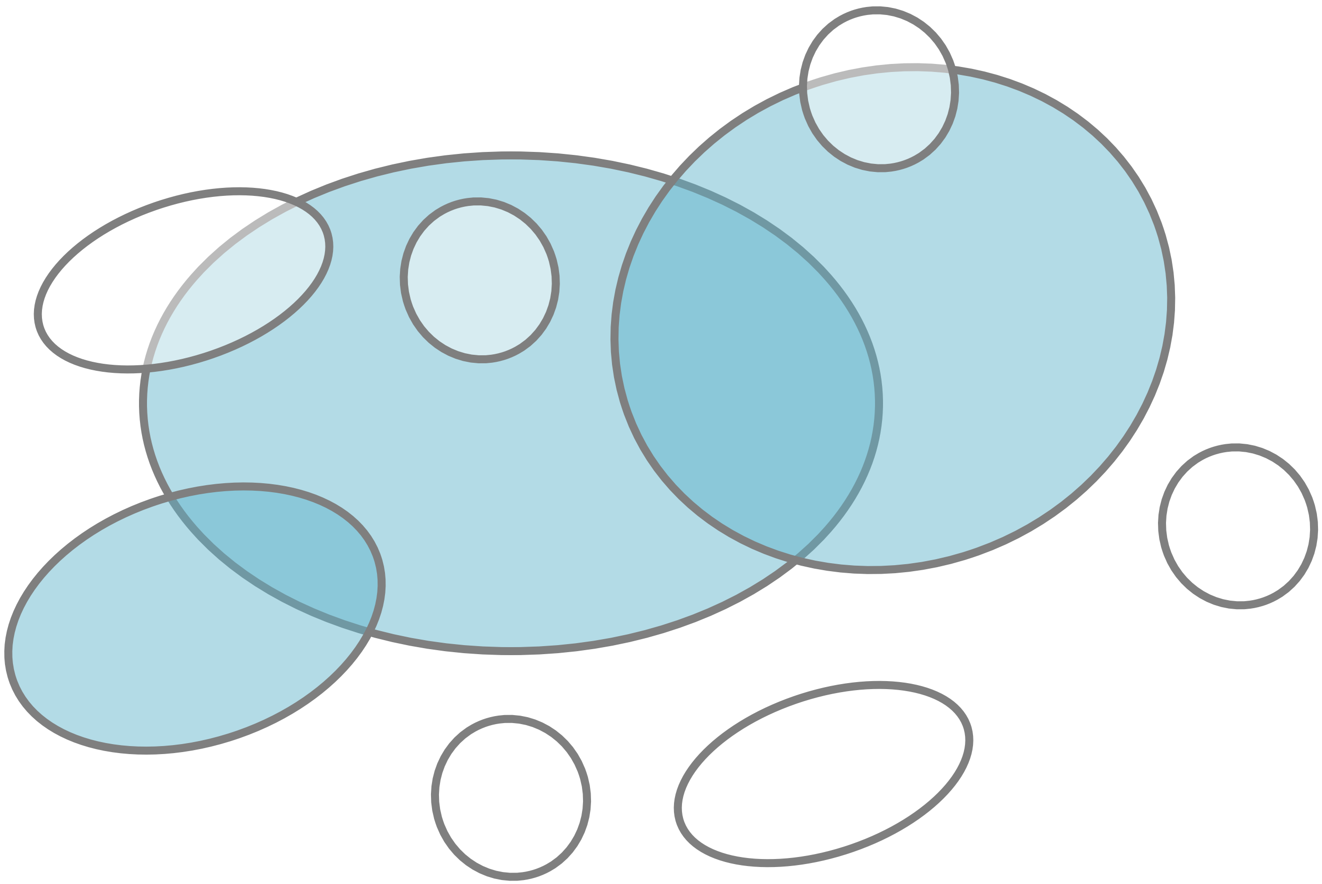
Simple Greedy Heuristic

- **Goal: Maximize the covered area**



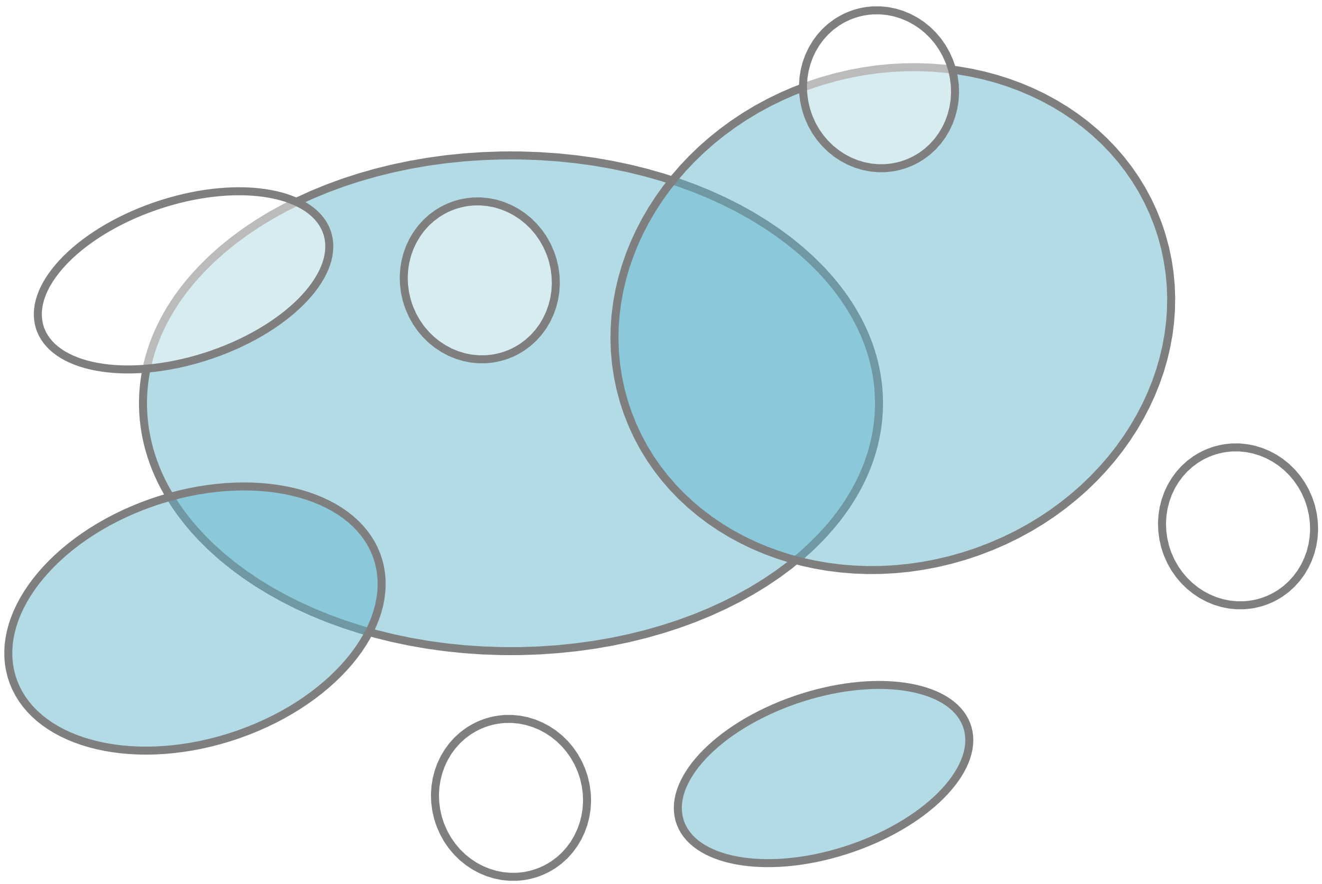
Simple Greedy Heuristic

- **Goal: Maximize the covered area**

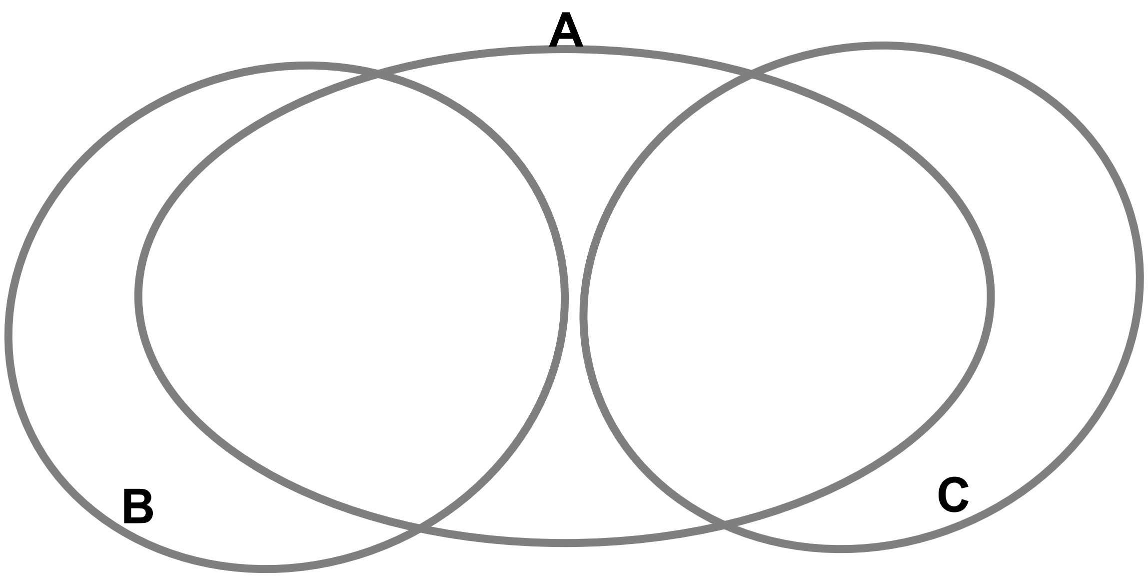


Simple Greedy Heuristic

- **Goal: Maximize the covered area**



When Greedy Heuristic Fails?



- **Goal:** Maximize the size of the covered area
- Greedy first picks A and then C
- But the optimal way would be to pick B and C

Approximation Guarantee

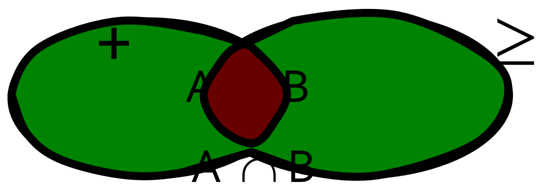
- **<u>Greedy</u> produces a solution *A***
  where: $F(A) \geq (1\text{-}1/e)*OPT$ $(F(A) \geq 0.63*OPT)$
  [Nemhauser, Fisher, Wolsey '78]
- **Claim holds for functions $F(\cdot)$ with 2 properties:**
  - ***F* is monotone:** (adding more docs doesn't decrease coverage)
    if **A** $\subseteq$ ***B*** then $F(A) \leq F(B)$ and $F(\{\})=0$
  - ***F* is submodular:**
    adding an element to a set gives less improvement than adding it to one of its subsets

Submodularity: Definition

Definition:

- Set function $F(\cdot)$ is called **submodular** if:
  For all $A,B \subseteq W$:

$$F(A) + F(B) \geq F(A \cup B) + F(A \cap B)$$



+

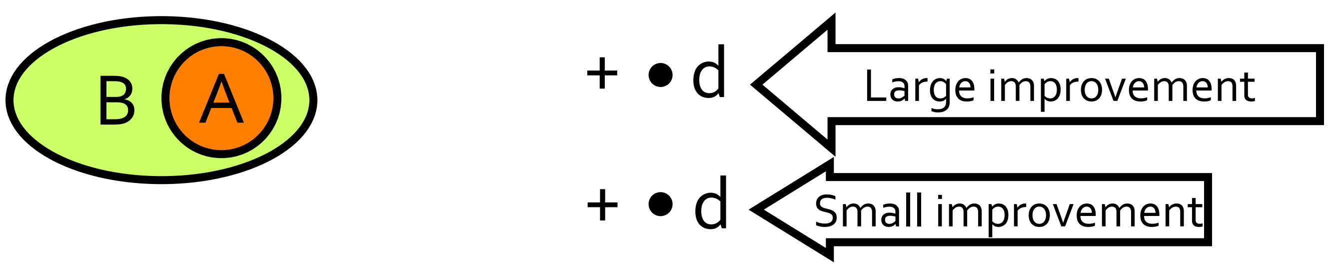
Submodularity: Or equivalently

- **Diminishing returns characterization**

Equivalent definition:

- Set function ***F(·)*** is called **submodular** if:
  For all $\boldsymbol{A} \subseteq \boldsymbol{B}$:

$$\underbrace{F(A \cup \{d\}) - F(A)}_{\text{Gain of adding } d \text{ to a small set}} \geq \underbrace{F(B \cup \{d\}) - F(B)}_{\text{Gain of adding } d \text{ to a large set}}$$

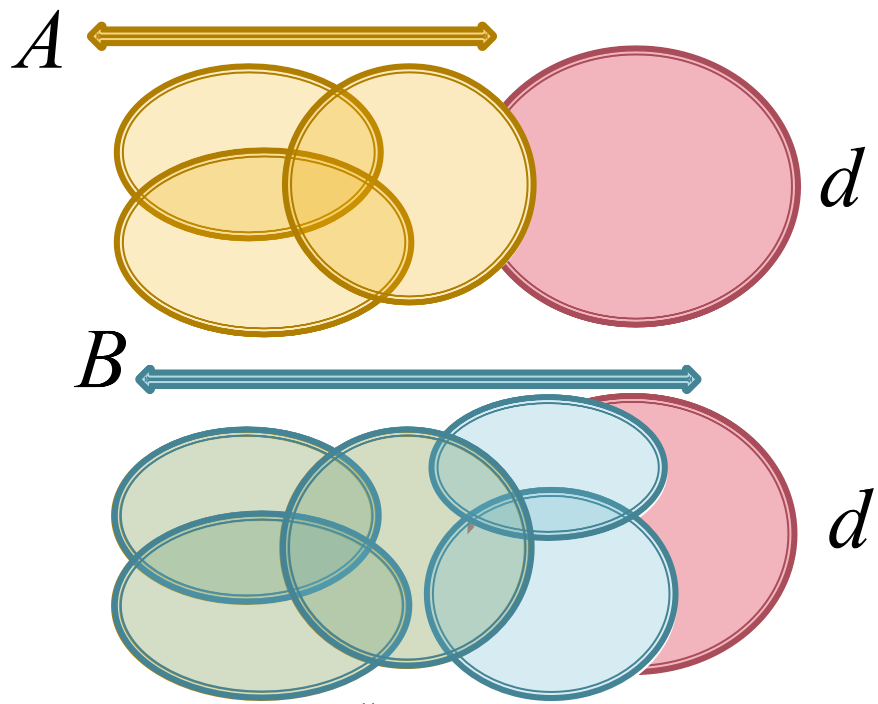


Example: Set Cover

- ***F(·)*** is **submodular**: $A \subseteq B$

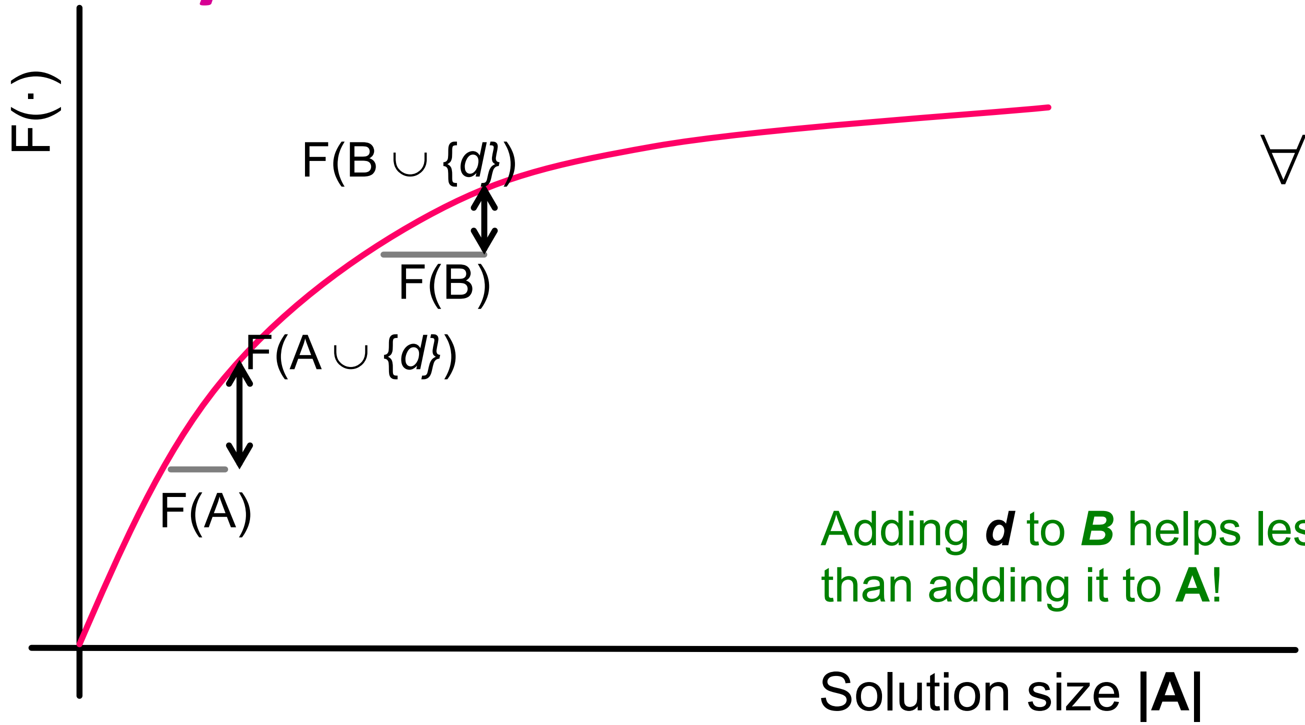
$$\underbrace{F(A \cup \{d\}) - F(A)}_{\text{Gain of adding } \mathbf{d} \text{ to a small set}} \geq \underbrace{F(B \cup \{d\}) - F(B)}_{\text{Gain of adding } \mathbf{d} \text{ to a large set}}$$

- **Natural example:**
  - Sets $d_1, \ldots, d_m$
  - $F(A) = |\bigcup_{i \in A} d_i|$
    (size of the covered area)
  - <u>**Claim:**</u>
    $F(A)$ **is submodular!**



Submodularity– Diminishing returns

- **Submodularity is discrete analogue of concavity**



$\forall A \subseteq B$

Adding ***d*** to ***B*** helps less than adding it to **A**!

$$\underbrace{F(A \cup \{d\}) - F(A)}_{\text{Gain of adding } X_d \text{ to a small set}} \geq \underbrace{F(B \cup \{d\}) - F(B)}_{\text{Gain of adding } X_d \text{ to a large set}}$$

Submodularity & Concavity

- **Marginal gain:**

$$\Delta_F(d|A) = F(A \cup \{d\}) - F(A)$$

- **Submodular:**

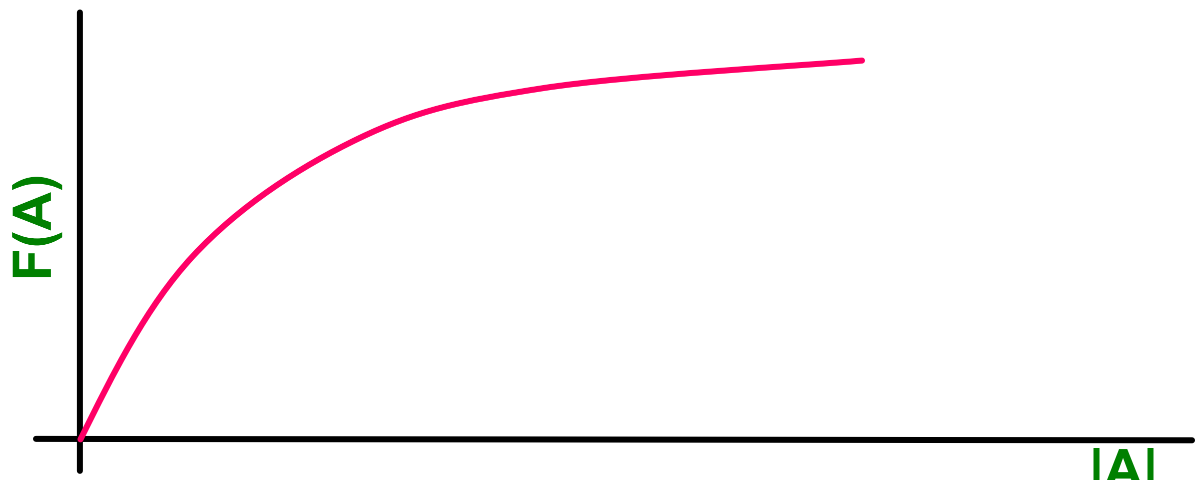
$$A \subseteq B$$

$$F(A \cup \{d\}) - F(A) \geq F(B \cup \{d\}) - F(B)$$

- **Concavity:**

$$a \leq b$$

$$f(a + d) - f(a) \geq f(b + d) - f(b)$$



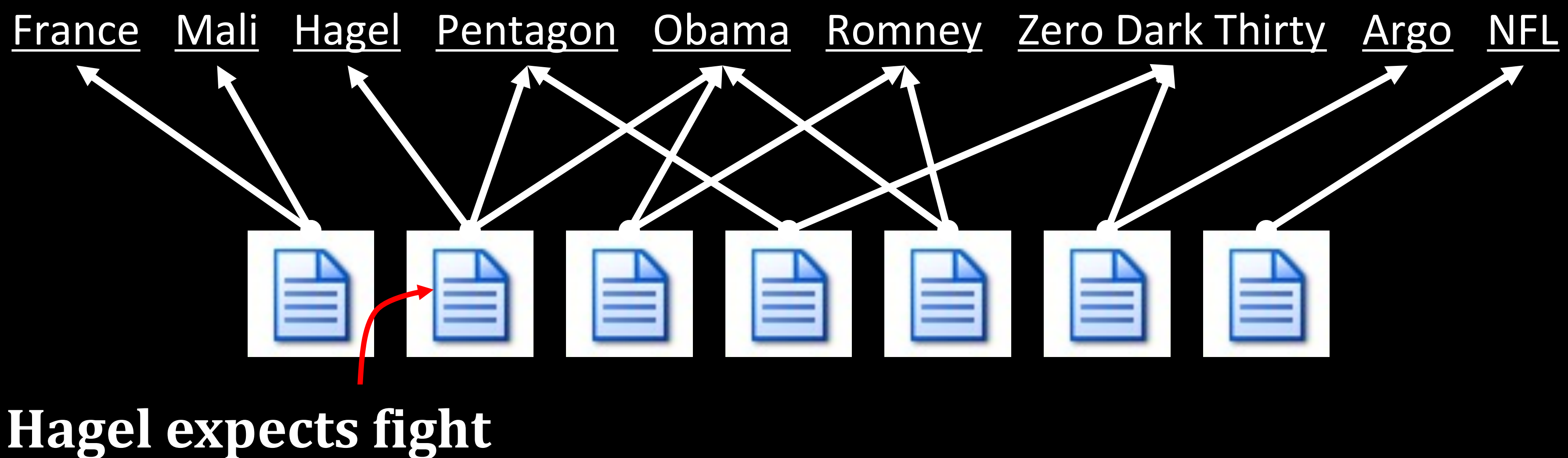
Submodularity: Useful Fact

- Let $F_1 \dots F_m$ be **submodular** and $\lambda_1 \dots \lambda_m > 0$ then $F(A) = \sum_{i=1}^{m} \lambda_i F_i(A)$ **is submodular**
  - **Submodularity is closed under non-negative linear combinations!**

- **This is an extremely useful fact:**
  - **Average of submodular functions is submodular:** $F(A) = \sum_i P(i) \cdot F_i(A)$
  - **Multicriterion optimization:** $F(A) = \sum_i \lambda_i F_i(A)$

Back to our problem

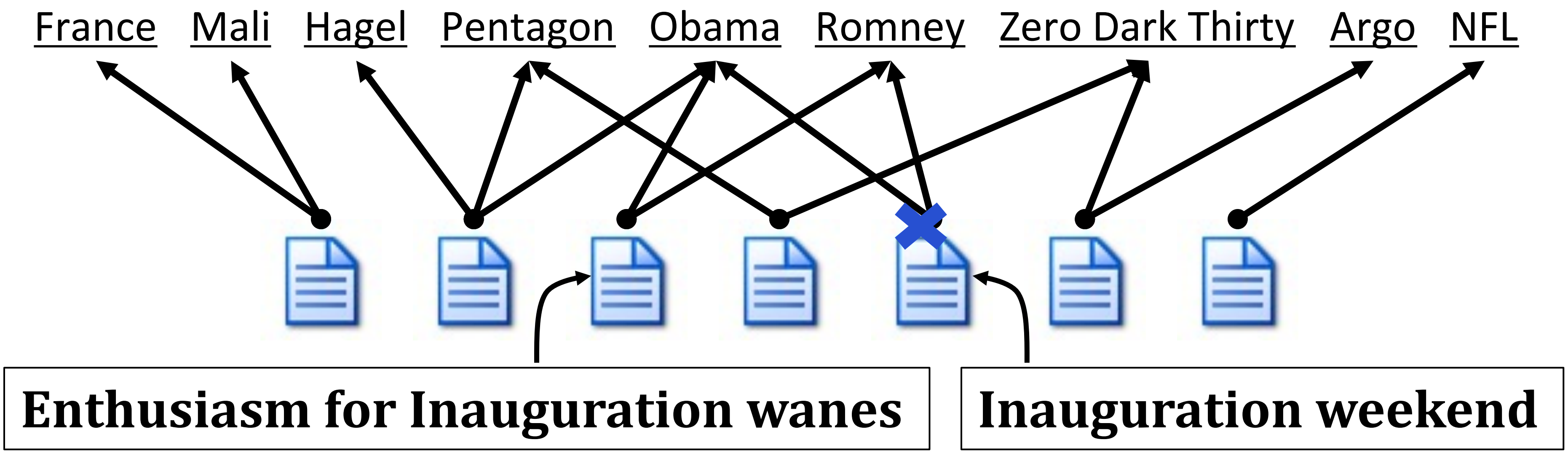
- **Q: What is being covered?**
- **A: Concepts** (In our case: Named entities)



- **Q: Who is doing the covering?**
- **A: Documents**

Back to our Concept Cover Problem

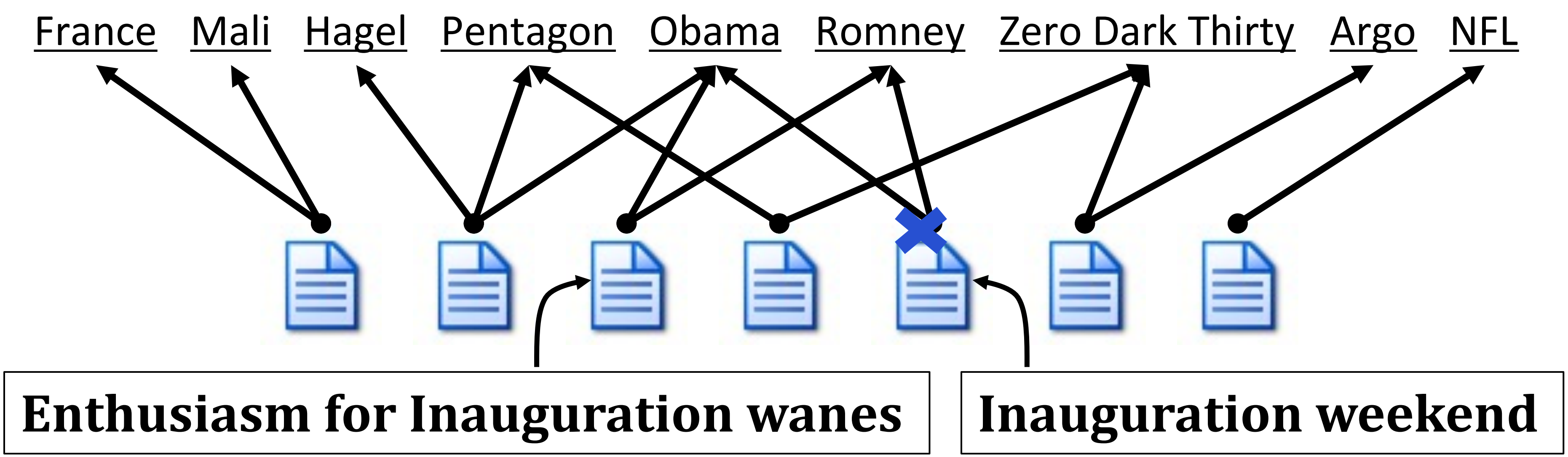
- **Objective: pick *k* docs that cover most concepts**



- ***F(A):* the number of concepts covered by *A***
  - ***Elements…concepts, Sets … concepts in docs***
  - ***F(A)*** is submodular and monotone!
  - We can use **greedy algorithm** to optimize ***F***

The Set Cover Problem

- **Objective:** pick ***k*** docs that cover most concepts



The good:

Penalizes redundancy

Submodular

The bad:

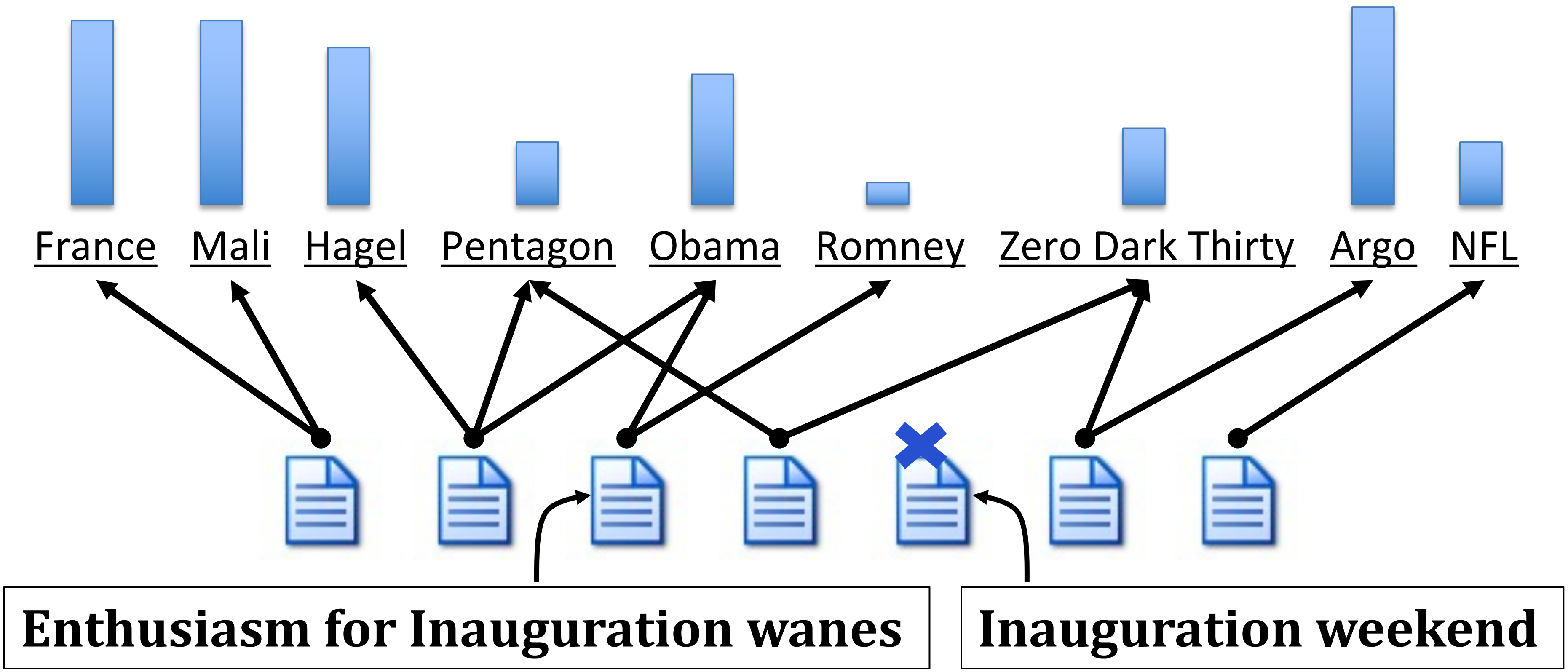
Concept importance?

All-or-nothing too harsh

Probabilistic Set Cover

Concept importance?

- **Objective:** pick ***k*** docs that cover most concepts



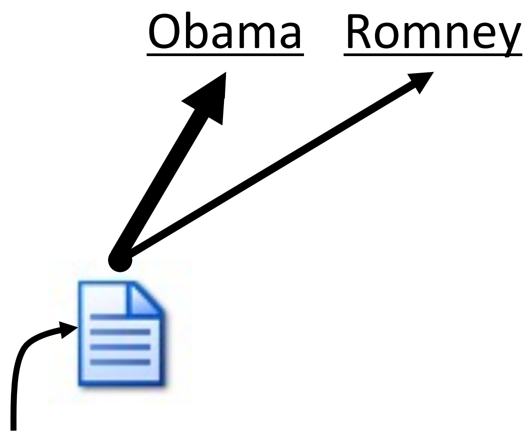
- **Each concept c has importance weight w_c**

All-or-nothing too harsh

- **Document coverage function**

$\mathrm{cover}_d(c) =$ **probability** document **d** covers concept **c**

[e.g., how strongly **d** covers **c**]



Enthusiasm for Inauguration wanes

Probabilistic Set Cover

- **Document coverage function:**

  $\text{cover}_d(c) =$ **probability** document **d** covers concept **c**

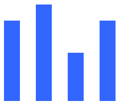
  - **Cover$_d$(c)** can also model how relevant is concept **c** for user **u**

- **Set coverage function:**

$$\text{cover}_{\mathcal{A}}(c) = 1 - \prod_{d \in \mathcal{A}} (1 - \text{cover}_d(c))$$

  - Prob. that at least one document in **A** covers **c**

- **Objective:**

concept weights 

$$\max_{\mathcal{A}:|\mathcal{A}| \leq k} F(\mathcal{A}) = \sum_c w_c \ \text{cover}_{\mathcal{A}}(c)$$

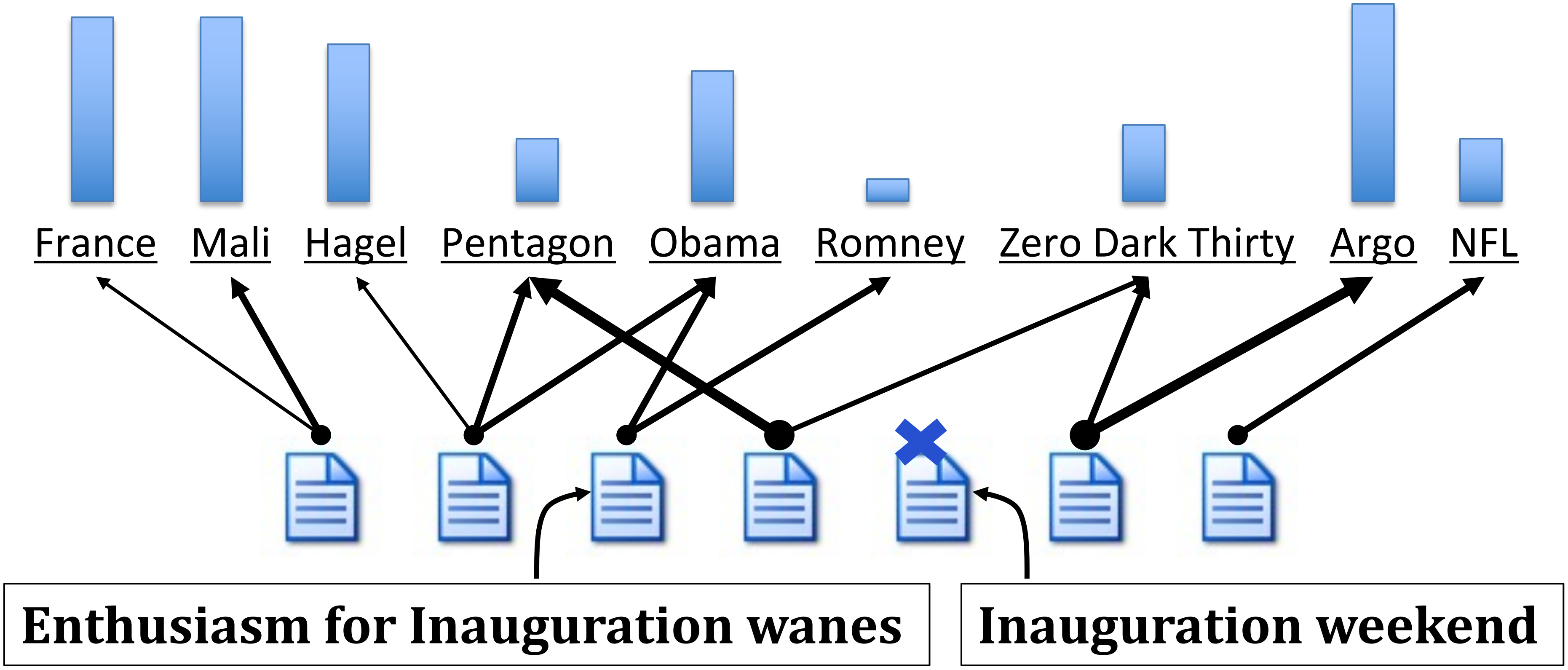
Optimizing F(A)

$$\max_{\mathcal{A}:|\mathcal{A}|\leq k} F(\mathcal{A}) = \sum_c w_c \ \mathrm{cover}_{\mathcal{A}}(c)$$

- The objective function is also **submodular**
  - Intuitively, it has a **diminishing returns** property
  - Greedy algorithm leads to a (1 – 1/e) ~ 63% approximation, i.e., a **near-optimal** solution

Summary: Probabilistic Set Cover

- **Objective:** pick ***k*** docs that cover most concepts



- Each concept c has importance weight w_c
- Documents partially cover concepts: $\mathbf{cover}_d(c)$

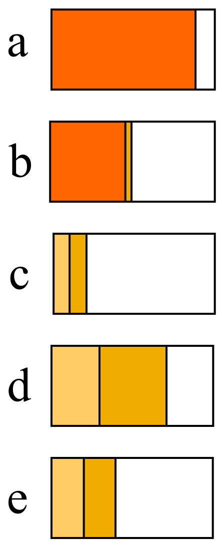
Lazy Optimization of Submodular Functions

Submodular Functions

Greedy

Marginal gain:

F(A∪x)-F(A)

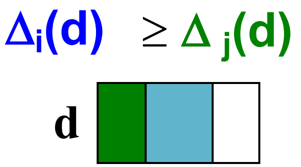


Add document with highest marginal gain

- **Greedy** algorithm is **slow!**
  - At each iteration we need to re-evaluate marginal gains of **all remaining documents**
  - **Runtime $O(|D| \cdot K)$ for selecting K documents out of the set of D of them**

Speeding up Greedy

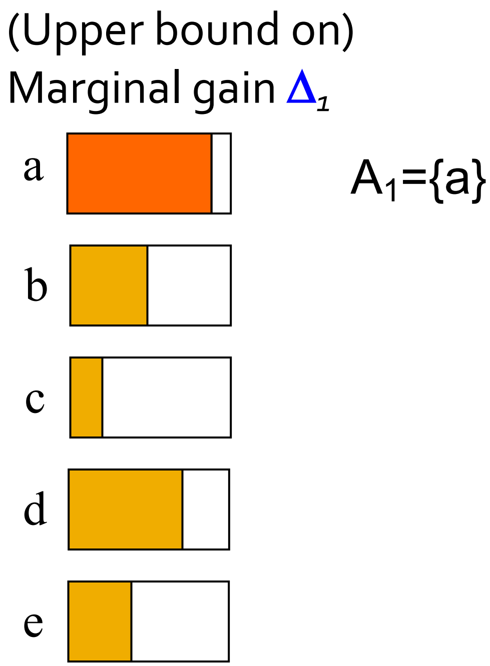
- **In round i:** So far we have $A_{i-1} = \{d_1, \dots, d_{i-1}\}$
  - Now we pick $\mathbf{d}_i = \arg\max_{d \in V} F(A_{i-1} \cup \{d\}) - F(A_{i-1})$
    - Greedy algorithm **maximizes the "marginal benefit"**
      $\Delta_i(d) = F(A_{i-1} \cup \{d\}) - F(A_{i-1})$
- **By submodularity property:**
  $F(A_i \cup \{d\}) - F(A_i) \geq F(A_j \cup \{d\}) - F(A_j)$ for $i < j$
- **Observation: By submodularity:**
  For every $d \in D$
  $\Delta_i(d) \geq \Delta_j(d)$ for $i < j$ since $A_i \subseteq A_j$
- **Marginal benefits $\Delta_i(d)$ only shrink!**
  (as i grows)



Selecting document ***d*** in step ***i*** covers more words than selecting ***d*** at step ***j*** (j>i)

Lazy Greedy

- **Idea:**
  - Use Δ_i as upper-bound on Δ_j ($j > i$)
- **Lazy Greedy:**
  - Keep an ordered list of marginal benefits Δ_i from previous iteration
  - Re-evaluate Δ_i only for top element
  - Re-sort and prune

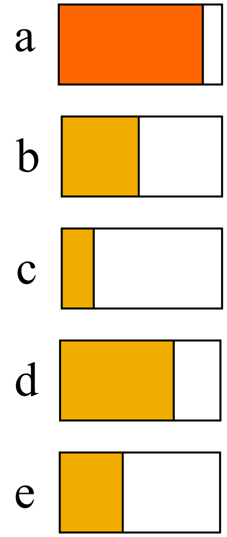


$$F(A \cup \{d\}) - F(A) \geq F(B \cup \{d\}) - F(B) \quad A \subseteq B$$

Lazy Greedy

- **Idea:**
  - Use Δ_i as upper-bound on Δ_j ($j > i$)
- **Lazy Greedy:**
  - Keep an ordered list of marginal benefits Δ_i from previous iteration
  - Re-evaluate Δ_i only for top element
  - Re-sort and prune

Upper bound on Marginal gain Δ_2



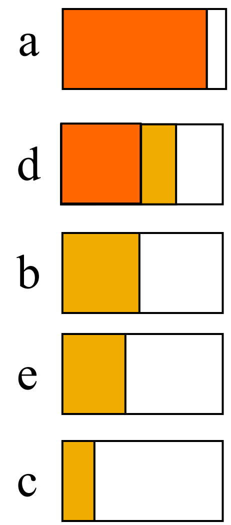
$A_1=\{a\}$

$$F(A \cup \{d\}) - F(A) \geq F(B \cup \{d\}) - F(B) \quad A \subseteq B$$

Lazy Greedy

- **Idea:**
  - Use Δ_i as upper-bound on Δ_j ($j > i$)
- **Lazy Greedy:**
  - Keep an ordered list of marginal benefits Δ_i from previous iteration
  - Re-evaluate Δ_i only for top element
  - Re-sort and prune

Upper bound on Marginal gain Δ_2



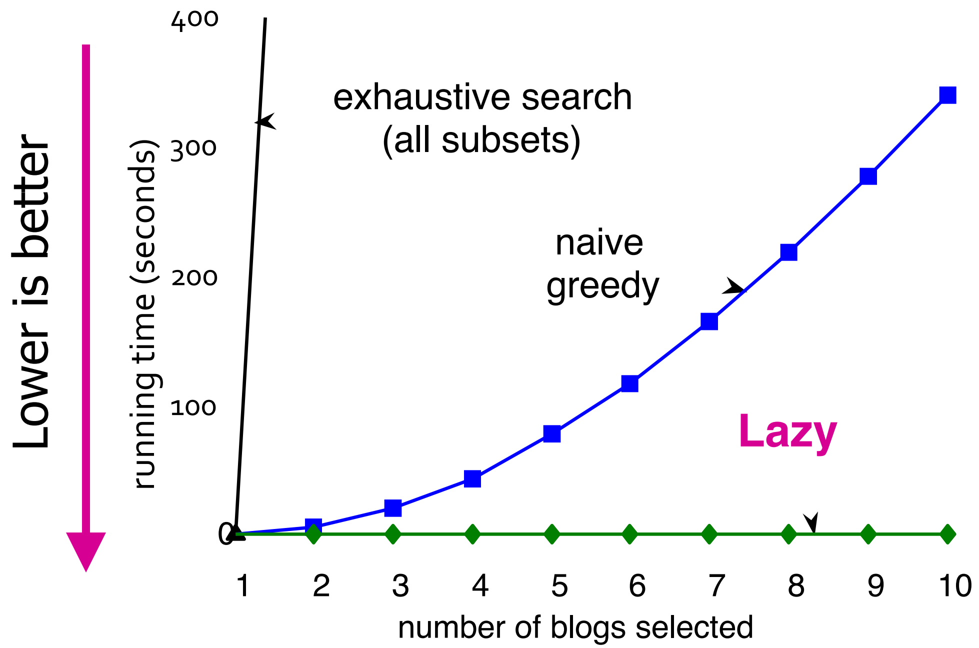
$A_1=\{a\}$

$A_2=\{a,b\}$

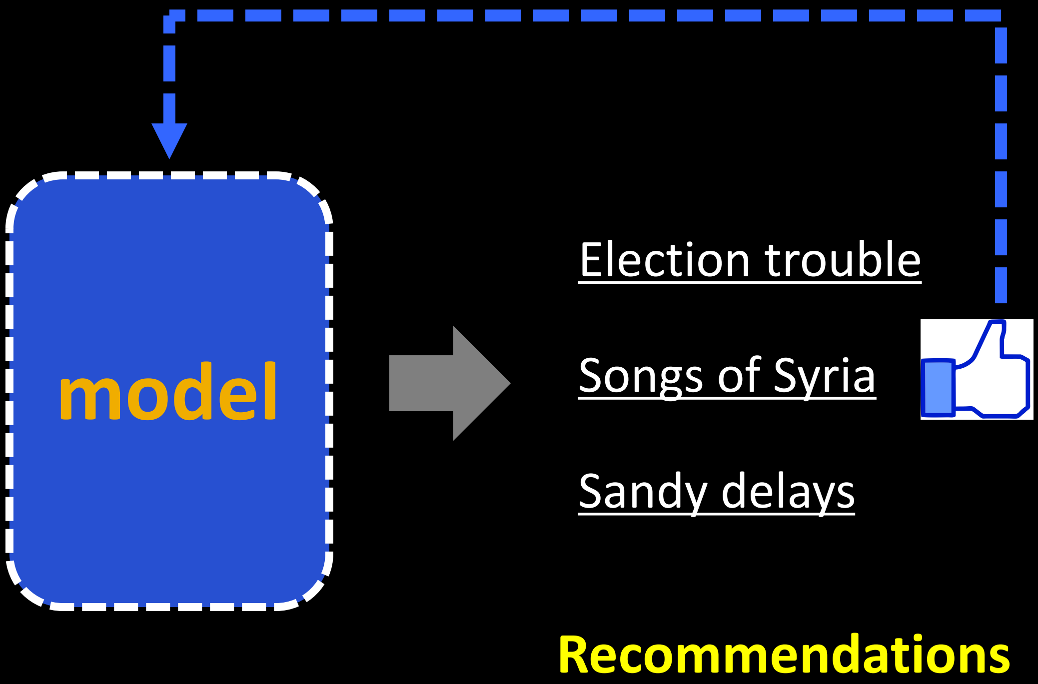
$$F(A \cup \{d\}) - F(A) \geq F(B \cup \{d\}) - F(B) \quad A \subseteq B$$

Summary so far

- **Summary so far:**
  - Diversity can be formulated as a set cover
  - Set cover is submodular optimization problem
  - Can be (approximately) solved using greedy algorithm
  - Lazy-greedy gives significant speedup

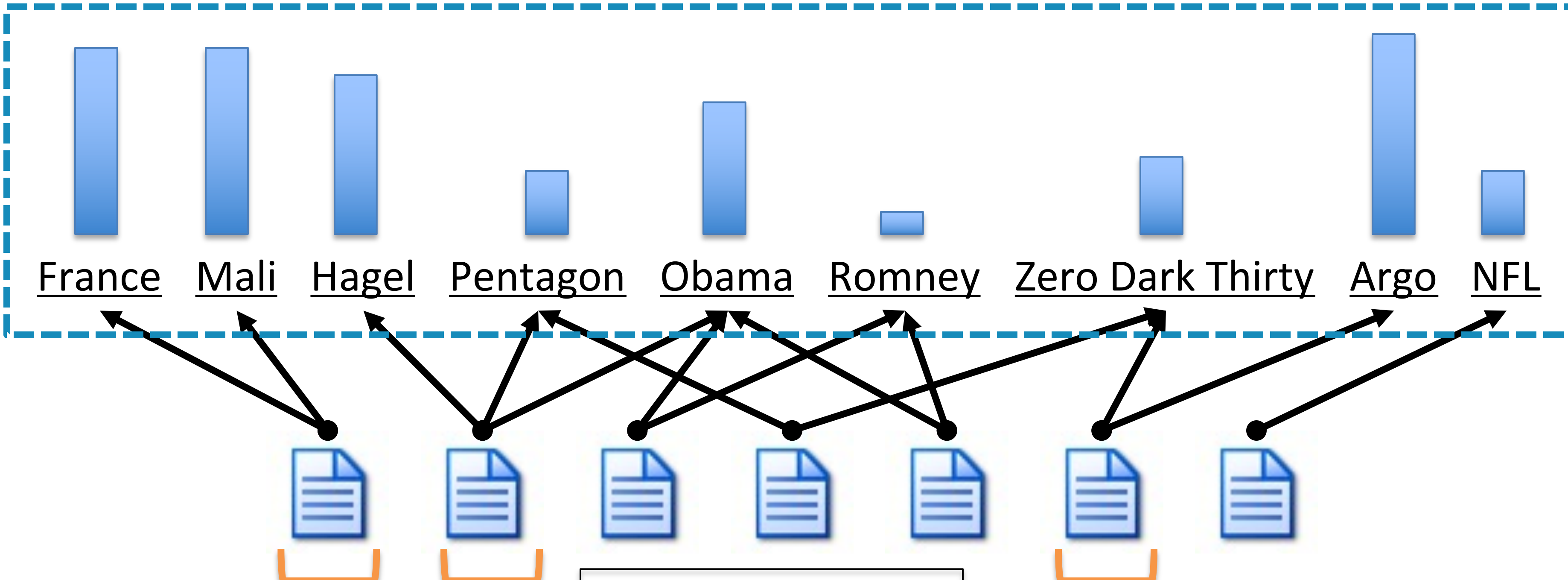


But what about **personalization**?



Concept Coverage

We assumed same concept weighting for all users



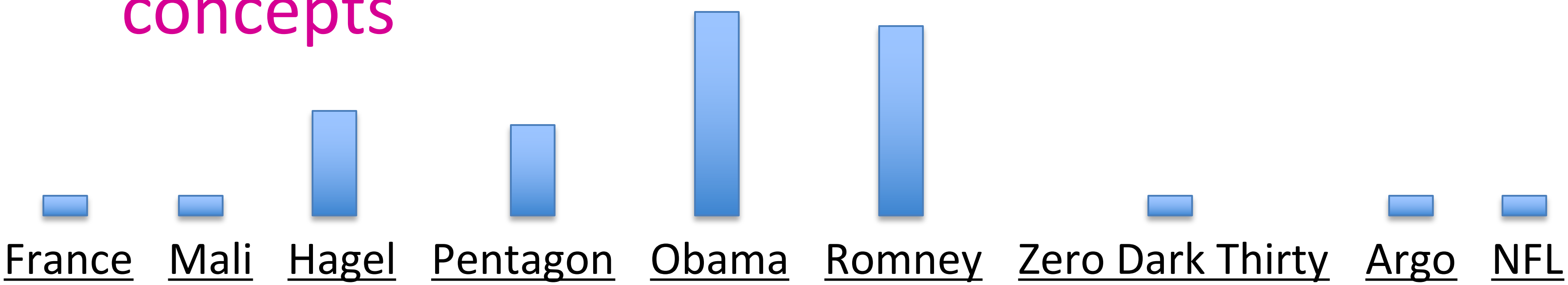
France intervenes

Chuck for Defense

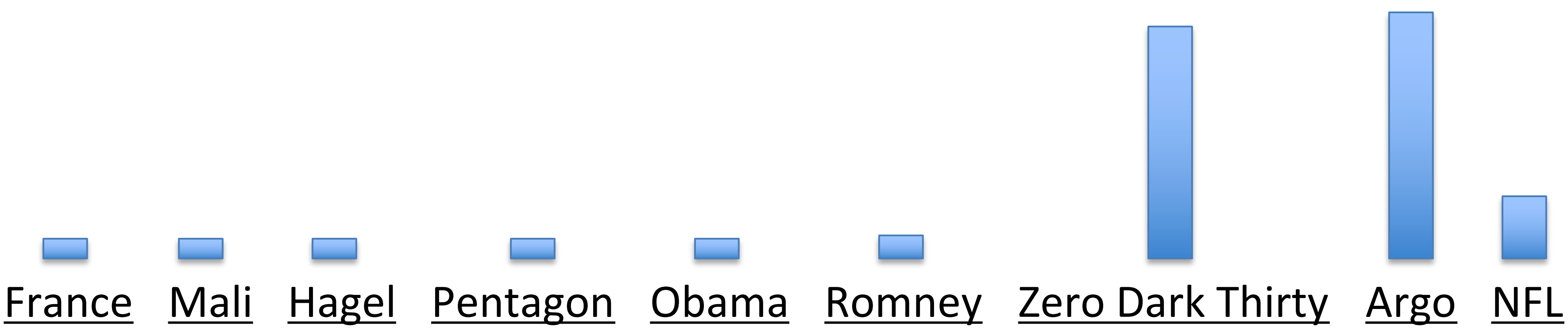
Argo wins big

Personal Concept Weights

- Each user has **different** preferences over concepts



politico



movie buff

Personal concept weights

- Assume each user ***u*** has **different** preference vector $\boldsymbol{w_c^{(u)}}$ over concepts ***c***

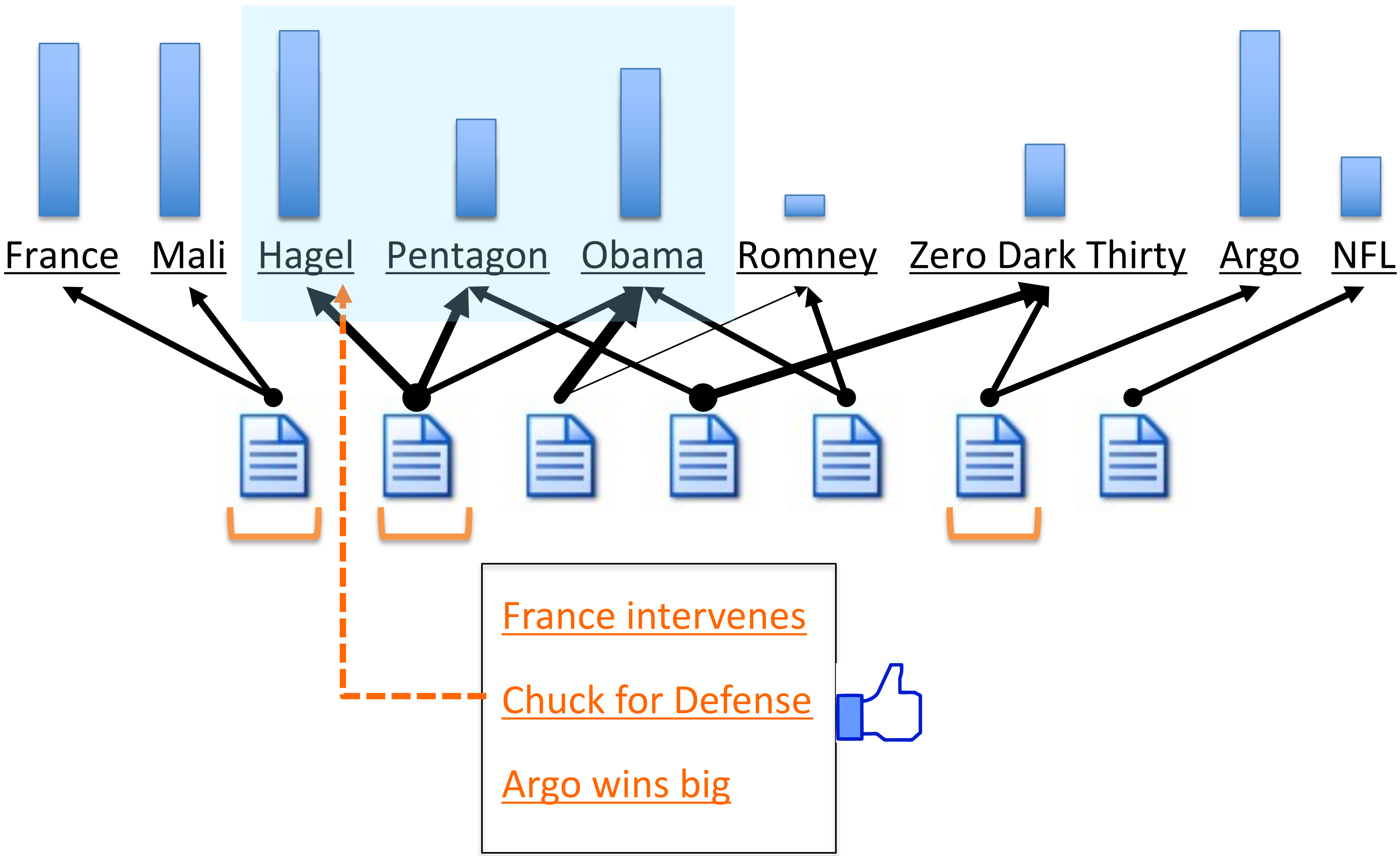
$$\max_{\mathcal{A}:|\mathcal{A}|\leq k} F(\mathcal{A}) = \sum_c w_c \; \text{cover}_{\mathcal{A}}(c)$$

$$\Downarrow$$

$$\max_{\mathcal{A}:|\mathcal{A}|\leq k} F(\mathcal{A}) = \sum_c w_c^{(u)} \; \text{cover}_{\mathcal{A}}(c)$$

- **Goal:** Learn personal concept weights from user feedback

Interactive Concept Coverage



Multiplicative Weights (MW)

- **Multiplicative Weights** algorithm
  - Assume each concept $\boldsymbol{c}$ has weight $\boldsymbol{w_c}$
  - We recommend document $\boldsymbol{d}$ and receive feedback, say $\boldsymbol{r} =$ **+1** or **-1**
  - **Update the weights:**
    - **For each** $\boldsymbol{c} \in \boldsymbol{X_d}$ **set** $\boldsymbol{w_c = \beta^r w_c}$
      - If concept **c** appears in doc **d** and we received positive feedback **r=+1** then we increase the weight $\mathbf{w_c}$ by multiplying it by $\boldsymbol{\beta}$ ($\boldsymbol{\beta > 1}$) otherwise we decrease the weight (divide by $\boldsymbol{\beta}$)
    - **Normalize weights so that** $\boldsymbol{\sum_c w_c = 1}$

Summary of the Algorithm

- **Steps of the algorithm:**
  1. Identify **items** to recommend from
  2. Identify **concepts** [what makes items redundant?]
  3. **Weigh** concepts by general importance
  4. Define **item-concept coverage function**
  5. **Select** items using probabilistic set cover
  6. Obtain **feedback**, **update** weights

Mining Massive Datasets: Conclusion

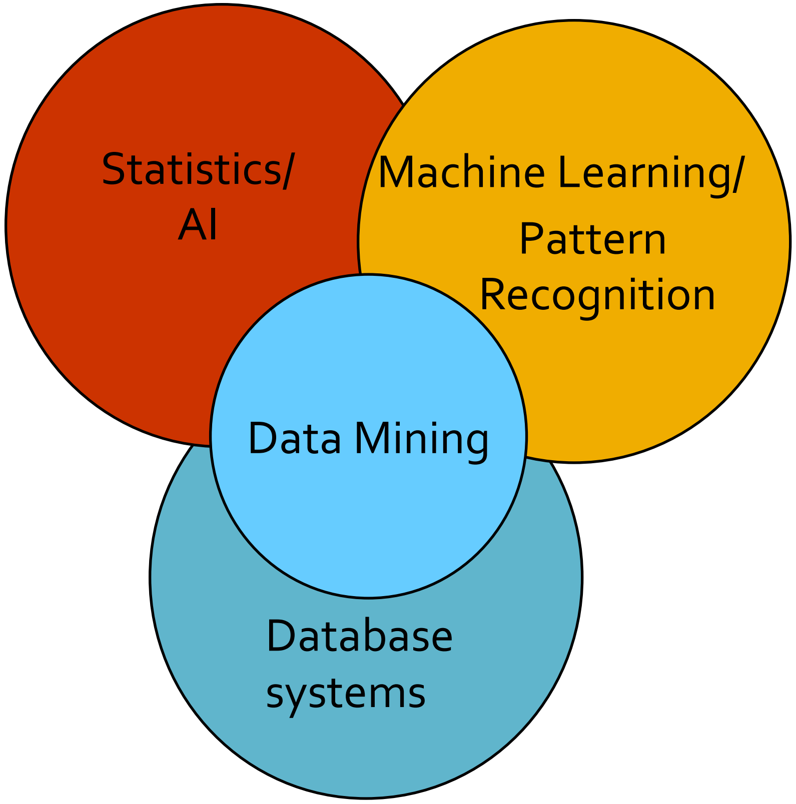
CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu

Data Mining

- **Models** **and** **tools for discovering patterns** **and** **answering queries** **that are:**
  - **Valid:** Hold on new data with some certainty
  - **Useful:** Should be possible to act on the item
  - **Unexpected:** Non-obvious to the system
  - **Understandable:** Humans should be able to interpret the pattern

Mining Massive Datasets

- **Overlaps with machine learning, statistics, artificial intelligence, databases, but more stress on**
  - **Scalability** of number of features and instances
  - **Algorithms** and **architectures**
  - Automation for handling **large data**



What We Have Covered

- Apriori
- MapReduce
- Association rules
- Frequent itemsets
- PCY
- Recommender systems
- PageRank
- TrustRank
- HITS
- Node2Vec
- Decision Trees
- GNN
- Web Advertising
- DGIM
- Bandits
- BFR
- Regret
- LSH
- MinHash
- SVD
- Clustering
- Matrix factorization
- CUR
- Bloom filters
- CURE
- Submodularity
- SGD
- Collaborative Filtering
- SimRank
- Random hyperplanes
- AND-OR constructions
- k-means
- Sketching
- Online Matching

How It All Fits Together

- **Based on different types of data:**
  - Data is **high dimensional**
  - Data is a **graph**
  - Data is **never-ending**
  - Data is **labeled**
- **Based on different models of computation:**
  - **Single machine in-memory**
  - **MapReduce**
  - **Streams**
  - **Batch (offline) vs. Active (online) algorithms**

How It All Fits Together

- Based on different applications:
  - **Recommender systems**
  - **Market basket analysis**
  - **Link analysis, spam detection**
  - **Duplicate detection** and **similarity search**
  - **Web advertising**
- Based on different "tools":
  - **Linear algebra:** SVD, Matrix factorization
  - **Optimization:** Stochastic gradient descent
  - **Dynamic programming:** Frequent itemsets
  - **Hashing:** LSH, Bloom filters

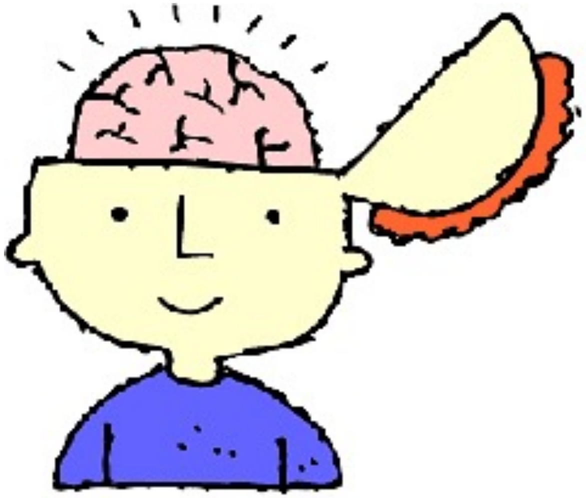
How It All Fits Together

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	Neural Networks	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Bandits	Duplicate document detection

In closing…

What we've learned this quarter

- MapReduce
- Association Rules
- Apriori algorithm
- Finding Similar Items
- Locality Sensitive Hashing
- Random Hyperplanes
- Dimensionality Reduction
- Singular Value Decomposition
- CUR method
- Clustering
- Recommender systems
- Collaborative filtering
- PageRank and TrustRank
- Hubs & Authorities
- k-Nearest Neighbors
- Perceptron
- Support Vector Machines
- Stochastic Gradient Descent
- Decision Trees
- Mining data streams
- Bloom Filters
- Flajolet-Martin
- Advertising on the Web



Map of Superpowers

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	Neural Networks	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Bandits	Duplicate document detection

Applying Your Superpowers

THE BIG PICTURE

- **How to analyze large datasets to discover models and patterns that are:**
  - **Valid:** Hold on new data with some certainty
  - **Novel:** Non-obvious to the system
  - **Useful:** Should be possible to act on the item
  - **Understandable:** Humans should be able to interpret the pattern

In Closing

- **You Have Done a Lot!!!**
- **And (hopefully) learned a lot!!!**
  - Answered questions and proved many interesting results
  - Implemented a number of methods

Thank You for the Hard Work!

(and good luck with the exam, and have a good break) ☺