TTIC 31190: Natural Language Processing

Kevin Gimpel

Winter 2016

Lecture 2: Text Classification

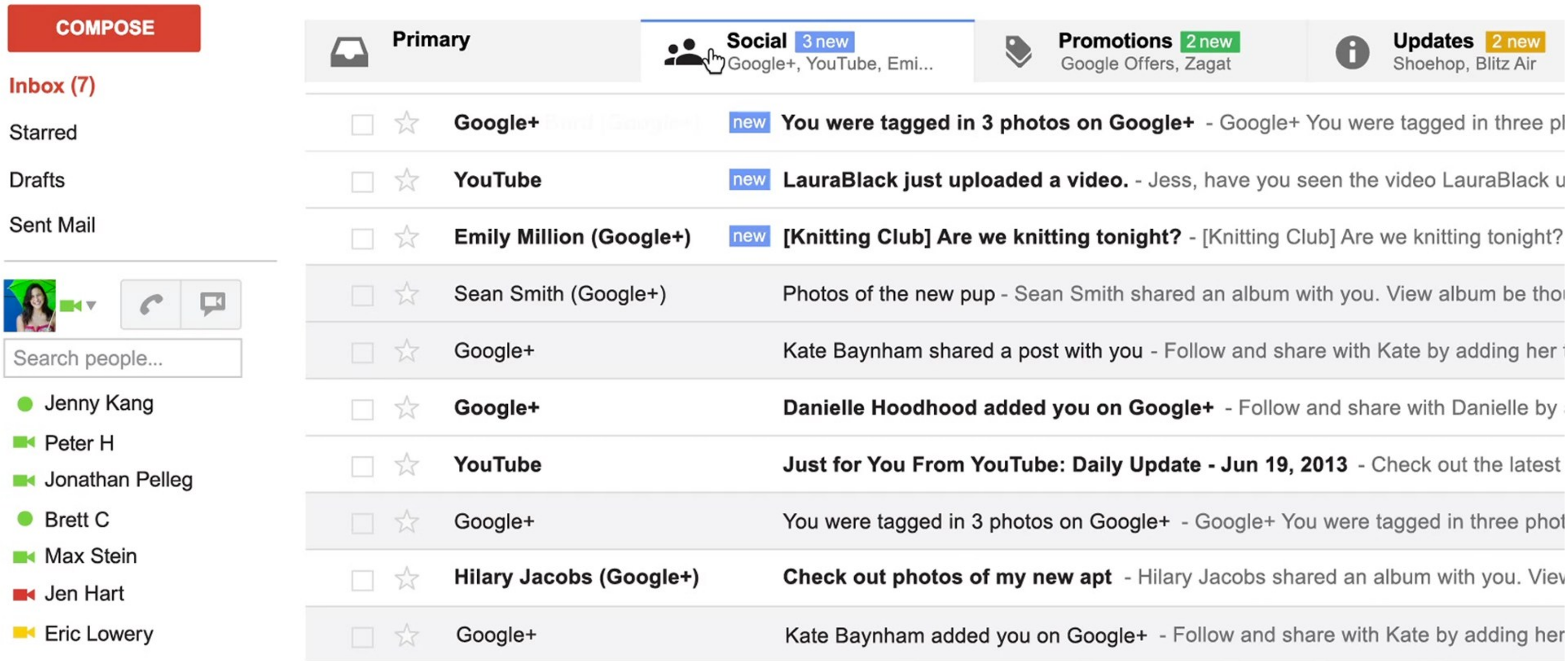
- Please email me (kgimpel@ttic.edu) with the following:
  - your name
  - your email address
  - whether you taking the class for credit

- I will use your address to create a mailing list for course announcements

Roadmap

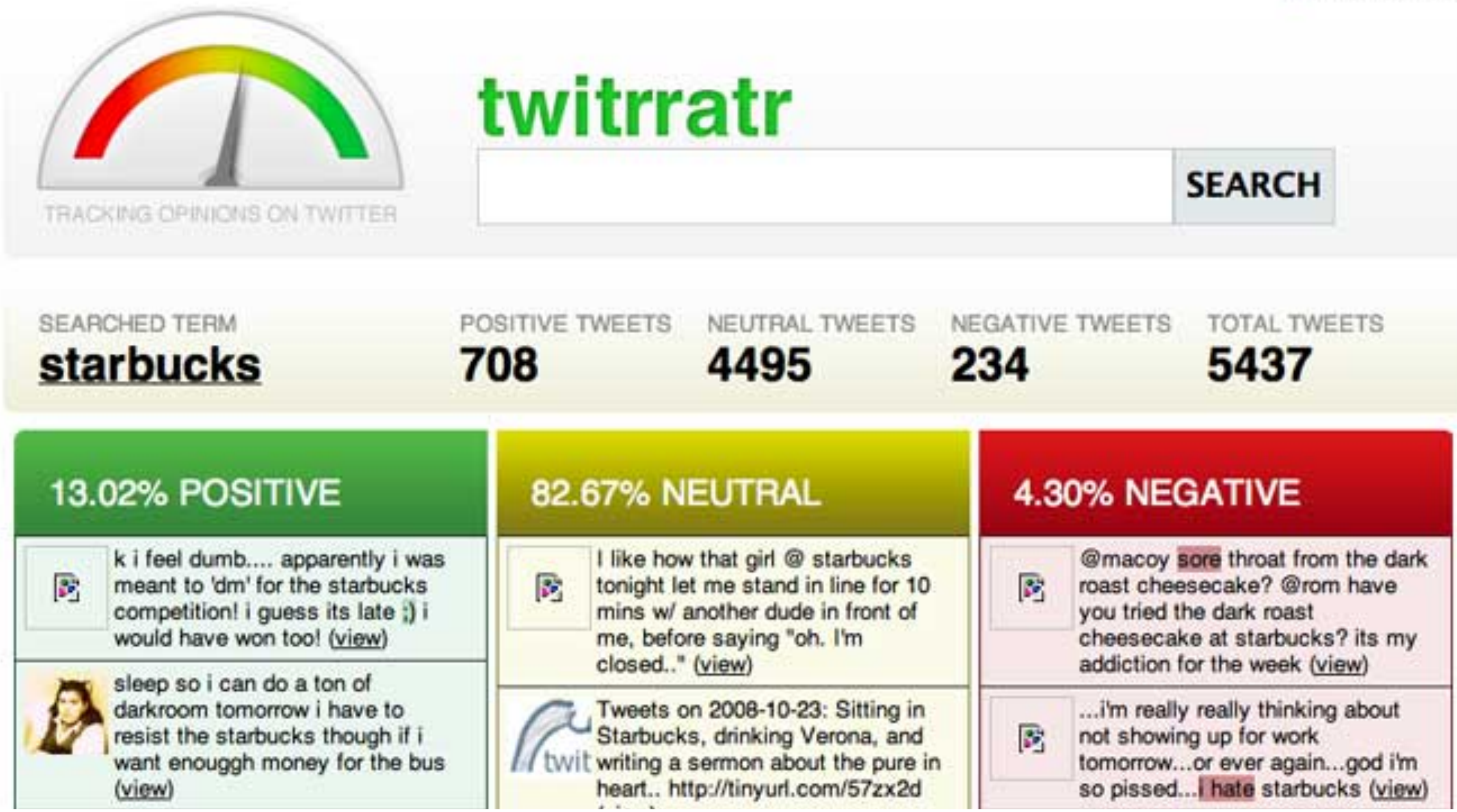
- classification
- words
- lexical semantics
- language modeling
- sequence labeling
- syntax and syntactic parsing
- neural network methods in NLP
- semantic compositionality
- semantic parsing
- unsupervised learning
- machine translation and other applications

Text Classification



- spam / not spam
- priority level
- category (primary / social / promotions / updates)

Sentiment Analysis



Classification

- datasets
- features
- learning

NLP Datasets

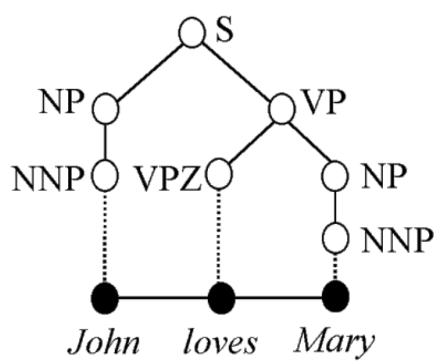
- NLP datasets include inputs (usually text) and outputs (usually some sort of annotation)

Annotation

- supervised machine learning needs labeled datasets, where labels are called ground truth
- in NLP, labels are annotations provided by humans
- there is always some disagreement among annotators, even for simple tasks
- these annotations are called a gold standard, not ground truth

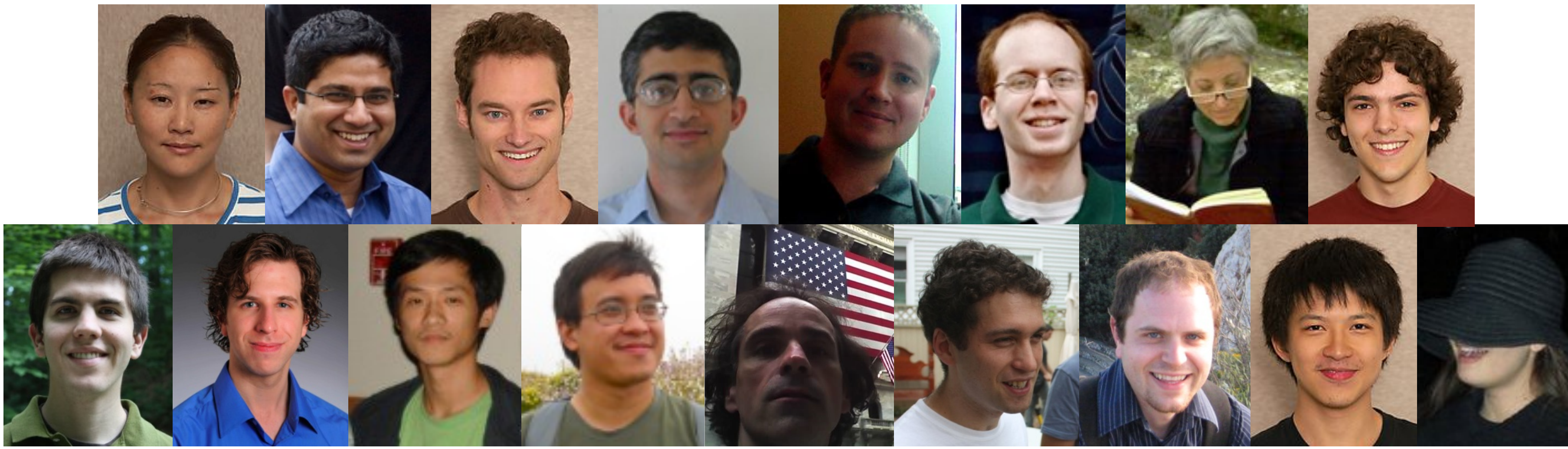
How are NLP datasets developed?

1. paid, trained human annotation
   - this is the traditional approach
   - researchers write annotation guidelines, recruit & pay annotators (often linguists)
   - more consistent annotations, but costly to scale
   - e.g., Penn Treebank (1993)
     - 1 million words, mostly Wall Street Journal, annotated with part-of-speech tags and syntactic parse trees



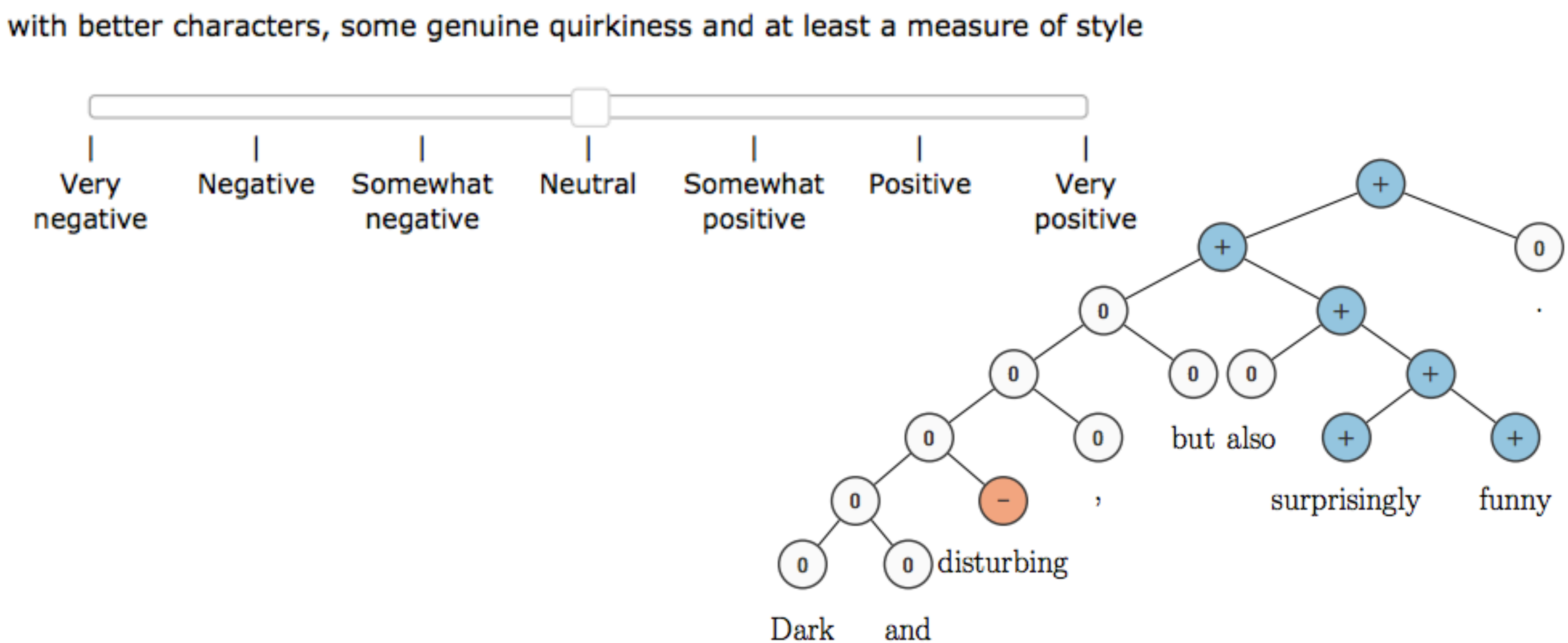
Example: Twitter part-of-speech annotation

17 CMU researchers annotated ~2000 tweets



Gimpel, Schneider, O'Connor, Das, Mills, Eisenstein, Heilman, Yogatama, Flanigan, Smith. "Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments," ACL 2011.

2. crowdsourcing
   - more recent trend
   - Amazon Mechanical Turk
   - can’t really train annotators, but easier to get multiple annotations for each input (which can then be averaged)
   - e.g., Stanford Sentiment Treebank:



3. naturally-occurring annotation

– long history: used by IBM for speech recognition and statistical machine translation

There's No Data Like More Data

• Dick Garwin's correspondence	~2.5M words
• Associated Press	20M words
• Oil company	25M words
• Federal Register	??M words
• American Printing House for the Blind	60M words
• IBM Deposition	100M words
• Canadian Hansard English	100M words

credit: Brown & Mercer, 20 Years of Bitext Workshop, 2013

3. naturally-occurring annotation

– long history: used by IBM for speech recognition and statistical machine translation

There's No Data Like More Data

• Dick Garwin's correspondence	~2.5M words
• Associated Press	20M words
• Oil company	25M words
• Federal Register	??M words
• American Printing House for the Blind	60M words
• IBM Deposition	100M words
• Canadian Hansard English	100M words

credit: Brown & Mercer, 20 Years of Bitext Workshop, 2013

– how might you find naturally-occurring data for:

- conversational agents
- summarization
- coreference resolution

Annotator Agreement

- given annotations from two annotators, how should we measure inter-annotator agreement?

Annotator Agreement

- given annotations from two annotators, how should we measure inter-annotator agreement?
  - percent agreement?

Annotator Agreement

- given annotations from two annotators, how should we measure inter-annotator agreement?
  - percent agreement?
  - Cohen's Kappa (Cohen, 1960) accounts for agreement by chance
  - generalizations exist for more than two annotators (Fleiss, 1971)

Text Classification Data

- There are many annotated datasets
  - Stanford Sentiment Treebank: fine-grained sentiment analysis of movie reviews
  - subjectivity/objectivity sentence classification
  - binary sentiment analysis of customer reviews
  - TREC question classification

- Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	
solondz may well be the only one laughing at his own joke	
obstacles pop up left and right , as the adventure gets wilder and wilder .	

- Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective
solondz may well be the only one laughing at his own joke	
obstacles pop up left and right , as the adventure gets wilder and wilder .	

- Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective
solondz may well be the only one laughing at his own joke	subjective
obstacles pop up left and right , as the adventure gets wilder and wilder .	objective

- Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective
solondz may well be the only one laughing at his own joke	subjective
obstacles pop up left and right , as the adventure gets wilder and wilder .	objective

- How was this dataset generated?

- Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective
solondz may well be the only one laughing at his own joke	subjective
obstacles pop up left and right , as the adventure gets wilder and wilder .	objective

- How was this dataset generated?
  – IMDB plot summaries: objective
  – Rotten Tomatoes snippets: subjective

- Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective
solondz may well be the only one laughing at his own joke	subjective
obstacles pop up left and right , as the adventure gets wilder and wilder .	objective

- How might you generate a dataset like this?

- customer review sentiment classification:

it works with a minimum of fuss .	
size - bigger than the ipod	
i 've had this thing just over a month and the headphone jack has already come loose .	
you can manage your profile , change the contrast of backlight , make different type of display , either list or tabbed .	
i replaced it with a router raizer and it works much better .	

- customer review sentiment classification:

it works with a minimum of fuss .	positive
size - bigger than the ipod	negative
i 've had this thing just over a month and the headphone jack has already come loose .	negative
you can manage your profile , change the contrast of backlight , make different type of display , either list or tabbed .	positive
i replaced it with a router raizer and it works much better .	negative

- question classification:

Who invented baseball ?	human
CNN is an acronym for what ?	abbreviation
Which Latin American country is the largest ?	location
How many small businesses are there in the U.S .	number
What would you add to the clay mixture to produce bone china ?	entity
What is the root of all evil ?	description

Classification

- datasets
- features
- learning

Classification Framework

inference: solve argmax

modeling: define score function

$$\mathrm{classify}(x, \boldsymbol{\theta}) = \operatorname*{argmax}_{y} \ \mathrm{score}(x, y, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

Classification Framework

inference: solve argmax

modeling: define score function

$$\text{classify}(x, \boldsymbol{\theta}) = \underset{y}{\text{argmax}}\ \ \text{score}(x, y, \boldsymbol{\theta})$$

learning: choose $\boldsymbol{\theta}$

our linear model text classifier:

$$\text{classify}^{\text{linear}}_{\text{text}}(\boldsymbol{x}, \boldsymbol{\theta}) = \underset{y \in \mathcal{L}}{\text{argmax}} \ \sum_i \theta_i f_i(\boldsymbol{x}, y)$$

Features for NLP

- NLP datasets include inputs and outputs
- features are usually not included
- you have to define your own features

Features for NLP

- NLP datasets include inputs and outputs
- features are usually not included
- you have to define your own features
- contrast this with UCI datasets, which include a fixed-length dense feature vector for every instance

Features for NLP

- NLP datasets include inputs and outputs
- features are usually not included
- you have to define your own features
- contrast this with UCI datasets, which include a fixed-length dense feature vector for every instance
- in NLP, features are usually sparse

Unigram Binary Features

- two example features:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \land \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \land \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

where $\mathbb{I}[S] = 1 \text{ if } S \text{ is true, } 0 \text{ otherwise}$

Unigram Binary Features

- two example features:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

where $\mathbb{I}[S] = 1 \text{ if } S \text{ is true, } 0 \text{ otherwise}$

- we usually think in terms of feature templates

Unigram Binary Features

- two example features:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

where $\mathbb{I}[S] = 1 \text{ if } S \text{ is true, } 0 \text{ otherwise}$

- we usually think in terms of feature templates
- unigram binary feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- to create features, this feature template is instantiated for particular labels and words

Higher-Order Binary Feature Templates

unigram binary template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

bigram binary template:

$$f^{\mathrm{b,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{``word1 word2''}]$$

trigram binary features

…

Unigram **Count** Features

- a ``count'' feature returns the count of a particular word in the text
- unigram count feature template:

$$f^{\mathrm{u,c}}(\boldsymbol{x}, y) = \begin{cases} \sum_{i=1}^{|\boldsymbol{x}|} \mathbb{I}[x_i = \ \textit{word}], & \text{if } \mathbb{I}[y = \ \text{label}] \\ 0, & \text{otherwise} \end{cases}$$

Feature Count Cutoffs

- problem: some features are extremely rare
- solution: only keep features that appear at least *k* times in the training data

Feature Count Cutoffs (Example)

- consider the following training dataset:

a great movie !	positive
not such a great movie	negative

- with the following single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- which features would remain in the model with a feature count cutoff of 2?

Feature Count Cutoffs (Example)

- consider the following training dataset:

a great movie !	positive
not such a great movie	negative

- with the following single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- which features would remain in the model with a feature count cutoff of 2?
  - none

Feature Count Cutoffs (Example)

- consider the following training dataset:

a great movie !	positive
not such a great movie	negative

- with the following single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- which features would remain in the model with a feature count cutoff of **1**?

Feature Count Cutoffs (Example)

- consider the following training dataset:

a great movie !	positive
not such a great movie	negative

- with the following single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \mathit{word}]$$

- which features would remain in the model with a feature count cutoff of **0**?

Classification

- datasets
- features
- learning
  - empirical risk minimization
  - surrogate loss functions
  - gradient-based optimization

Learning: Empirical Risk Minimization

- In a machine learning course, you learn about many different learning frameworks

Learning: Empirical Risk Minimization

- In a machine learning course, you learn about many different learning frameworks
- Since we have limited time, we will be greedy and focus on a single framework that maximizes

$$\alpha \, \text{ease\_of\_use} + \beta \, \text{effectiveness} + \gamma \, \text{applicability}$$

(for some positive constants α, β, γ)

We will start it today but continue to add to it later

Cost Functions

- cost function: scores outputs against a gold standard

$$\mathrm{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task
- usual conventions: $\mathrm{cost}(y, y) = 0$

$$\mathrm{cost}(y, y') = \mathrm{cost}(y', y)$$

Cost Functions

- cost function: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task

- for classification, what cost should we use?

Cost Functions

- cost function: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- should be as close as possible to the actual evaluation metric for your task

- for classification, what cost should we use?

$$\text{cost}(y, y') = \mathbb{I}[y \neq y']$$

- how about for other NLP tasks?

Risk Minimization

- given training data: $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$
  where each $y^{(i)} \in \mathcal{L}$ is a label
- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution $P(\boldsymbol{x}, y)$
- we want to solve the following:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{\theta})) \right]$$

Risk Minimization

- given training data: $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$
  where each $y^{(i)} \in \mathcal{L}$ is a label
- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution $P(\boldsymbol{x}, y)$
- we want to solve the following:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{\theta})) \right]$$

problem: *P* is unknown

Empirical Risk Minimization
(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta})) \right]$$

$$\Downarrow$$

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^{|\mathcal{T}|} \text{cost}(y^{(i)}, \text{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}))$$

Empirical Risk Minimization
(Vapnik et al.)

- replace expectation with sum over examples:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{\theta}))\right]$$

$$\downarrow$$

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}))$$

problem: NP-hard even for binary classification with linear models solution: replace "cost loss" (also called "0-1" loss) with a **surrogate** function that is easier to optimize

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}))$$

generalize to permit any loss function

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

solution: replace "cost loss" (also called "0-1" loss) with a **surrogate** function that is easier to optimize

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^{|\mathcal{T}|} \text{cost}(y^{(i)}, \text{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}))$$

generalize to permit any loss function

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_{i=1}^{|\mathcal{T}|} \text{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

Classification

- datasets
- features
- learning
  - empirical risk minimization
  - surrogate loss functions
  - gradient-based optimization

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

why is this so difficult to optimize?

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

why is this so difficult to optimize?
not necessarily continuous, can't use gradient-based optimization

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta})$$

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta})$$

this is continuous, but what are its drawbacks?

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta})$$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta})$$

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

max-score loss:

$$\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta})$$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta})$$

loss function underlying perceptron algorithm
(Rosenblatt, 1957-58)

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta})$$

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y'))$$

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta})$$

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y')\right)$$

loss function underlying support vector machines

Surrogate Loss Functions

cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{\theta}))$

perceptron loss:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta})$$

hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) + \text{cost}(y, y'))$$

hinge loss for our classification setting:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) + \delta\, \mathbb{I}[y \neq y'])$$

tunable hyperparameter

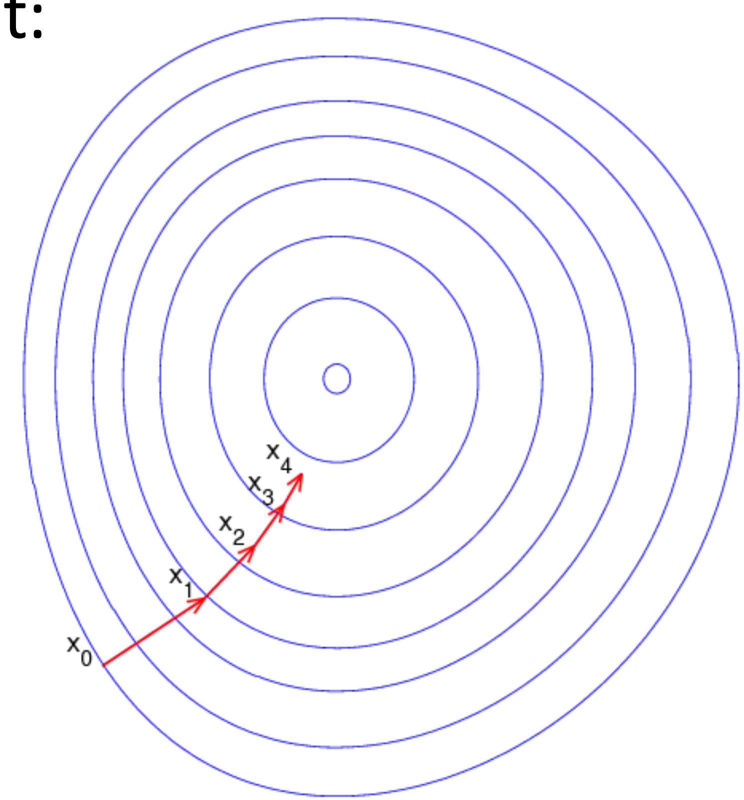
Classification

- datasets
- features
- learning
  - empirical risk minimization
  - surrogate loss functions
  - gradient-based optimization

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

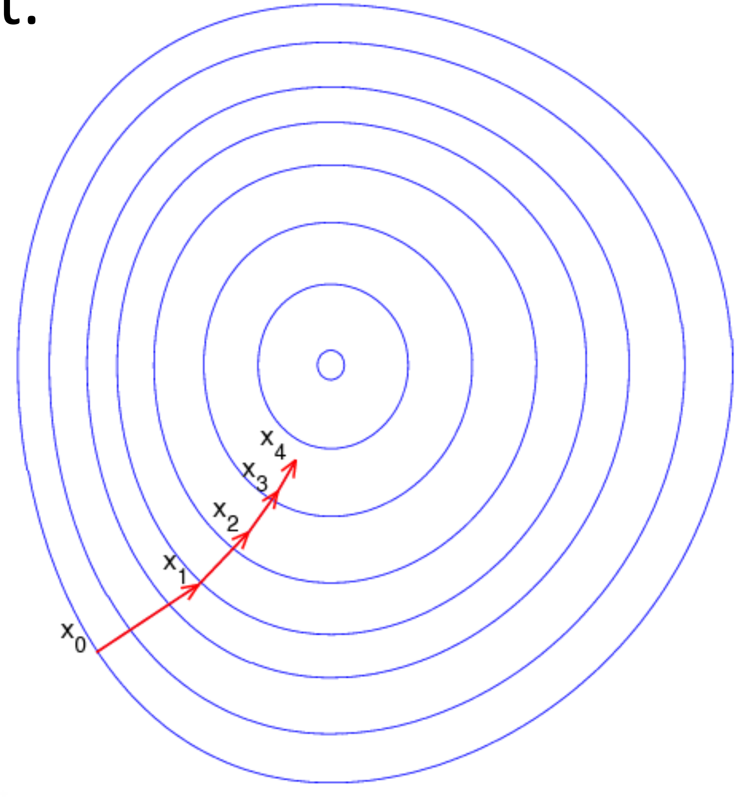
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$



Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize at iteration *t*

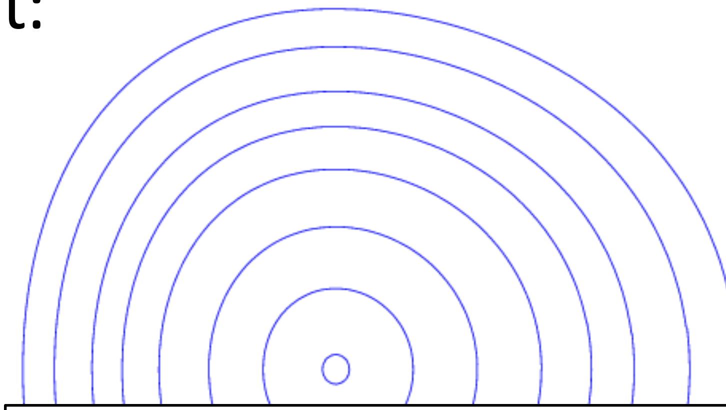
$\nabla F(\boldsymbol{\theta}^{(t)})$: gradient of objective function

- with conditions on stepsize and objective function, will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize at iteration *t*

$\boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$: gradient of objective function

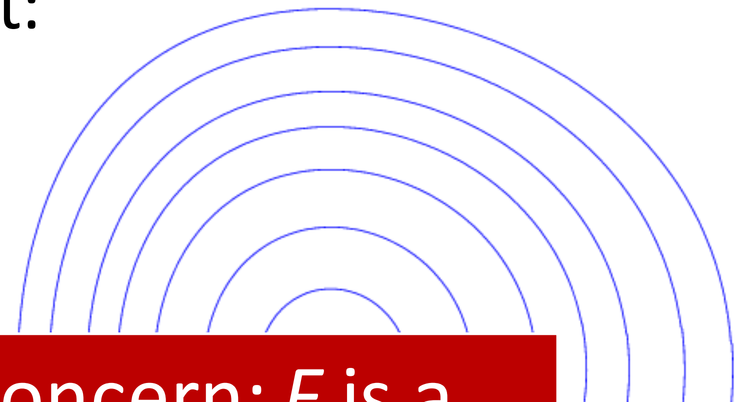
to speed convergence, can use line search to choose better stepsizes; also see L-BFGS

- with conditions on stepsize and objective function, will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize a

$\nabla F(\boldsymbol{\theta}^{(t)})$: grad

objective function

efficiency concern: *F* is a sum over all training

ence,

h to

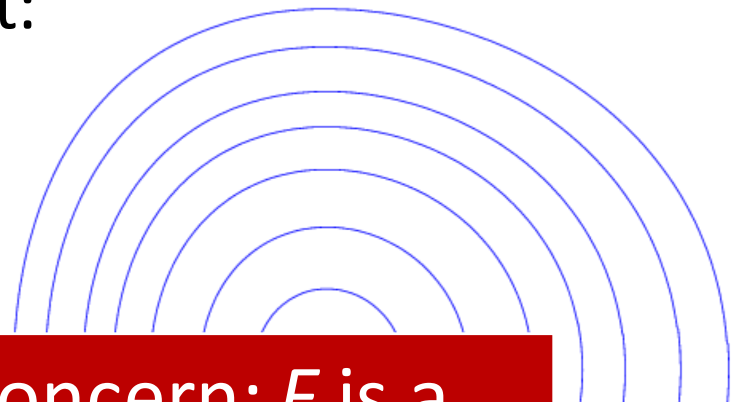
choose better stepsizes;

also see L-BFGS

- with conditions on stepsize and objective function, will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:



$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$$

$\eta^{(t)}$: stepsize a

$\boldsymbol{\nabla} F(\boldsymbol{\theta}^{(t)})$: grad
objective funct

efficiency concern: *F* is a sum over all training examples!

every parameter update requires iterating through

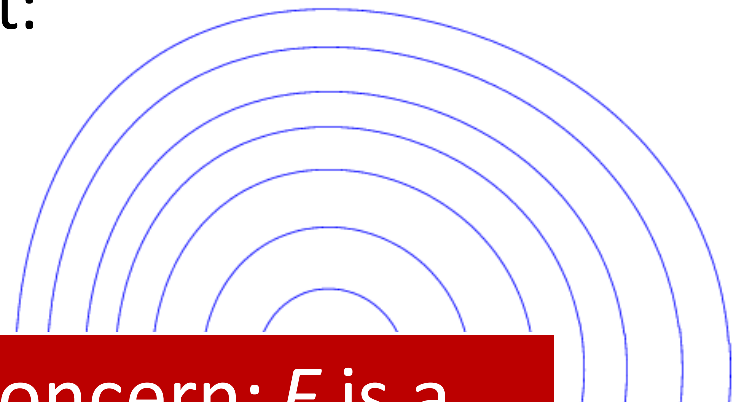
ence,
h to
osizes;

- with conditio on,
will converge to local minimum

Gradient Descent

- minimizes a function *F* by taking steps in proportion to the negative of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla F(\boldsymbol{\theta}^{(t)})$$



$\eta^{(t)}$: stepsize a

$\nabla F(\boldsymbol{\theta}^{(t)})$: grad
objective funct

efficiency concern: *F* is a sum over all training examples!

every parameter update requires iterating through entire training set

ence,
h to
sizes;

- with conditio … on,
will converge

Stochastic Gradient Descent

- applicable when objective function is a sum
- like gradient descent, except calculates gradient on a single example at a time ("online") or on a small set of examples ("mini-batch")

Stochastic Gradient Descent

- applicable when objective function is a sum
- like gradient descent, except calculates gradient on a single example at a time (“online”) or on a small set of examples (“mini-batch”)
- converges much faster than (batch) gradient descent
- with conditions on stepsize and objective function, will converge to local minimum
- there are many popular variants:

  SGD+momentum, AdaGrad, AdaDelta, Adam, RMSprop, etc.

What if *F* is not differentiable?

- some loss functions are not differentiable:

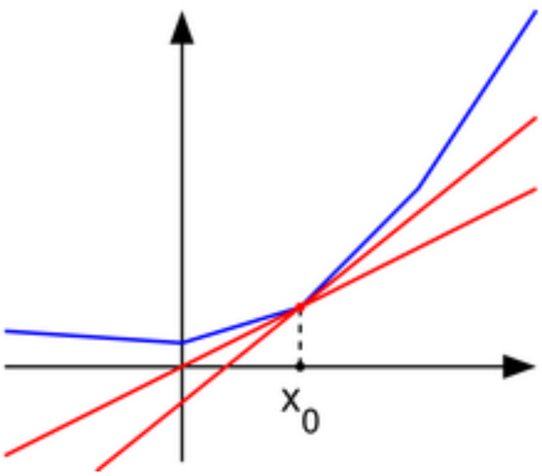
$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta})$$

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) + \delta \, \mathbb{I}[y \neq y'])$$

- but they *are* subdifferentiable, so we can compute subgradients and use (stochastic) subgradient descent

Subderivatives

- subderivative: generalization of derivative for nondifferentiable, convex functions



- there may be multiple subderivatives at a point (red lines)

- this set is called the subdifferential
- a convex function g is differentiable at point x_0 if and only if the subdifferential of g at x_0 contains only the derivative of g at x_0

Stochastic Subgradient Descent

- just like stochastic gradient descent, except replace gradients with subgradients
- similarly strong theoretical guarantees

Calculating Subgradients

- at points of differentiability, just use your rules for calculating gradients
- at points of nondifferentiability, just find a single subgradient; *any* subgradient will do
- e.g., max of convex functions (on board)

- Please email me (kgimpel@ttic.edu) with the following:
  - your name
  - your email address
  - whether you taking the class for credit

- I will use your address to create a mailing list for course announcements