Active Evaluation of Classifiers on Large Datasets

Namit Katariya, Arun Iyer, Sunita Sarawagi

IIT Bombay

December 13, 2012

International Conference on Data Mining, 2012

Problem setup

- Setup
  - A classifier $C(\mathbf{x})$ deployed on
  - A large unlabeled dataset D
- Estimate true accuracy μ of $C(\mathbf{x})$ on D
- Given
  - A labeled set L : small or unrepresentative of D
    - Measured accuracy on labeled set $\neq$ True accuracy on data
  - A human labeler
- Compelling problem in many real-life applications

Problem setup

- Setup
  - A classifier $C(\mathbf{x})$ deployed on
  - A large unlabeled dataset D
- Estimate true accuracy μ of $C(\mathbf{x})$ on D
- Given
  - A labeled set L : small or unrepresentative of D
    - Measured accuracy on labeled set $\neq$ True accuracy on data
  - A human labeler
- Compelling problem in many real-life applications

Problem setup

- Setup
  - A classifier $C(\mathbf{x})$ deployed on
  - A large unlabeled dataset D
- Estimate true accuracy μ of $C(\mathbf{x})$ on D
- Given
  - A labeled set L : small or unrepresentative of D
    - Measured accuracy on labeled set $\neq$ True accuracy on data
  - A human labeler
- Compelling problem in many real-life applications

Problem setup

- Setup
  - A classifier $C(\mathbf{x})$ deployed on
  - A large unlabeled dataset D
- Estimate true accuracy μ of $C(\mathbf{x})$ on D
- Given
  - A labeled set L : small or unrepresentative of D
    - Measured accuracy on labeled set $\neq$ True accuracy on data
  - A human labeler
- Compelling problem in many real-life applications

Our goal

1. Handle arbitrary classifier e.g. a user script
   - existing work assumes that $C(\mathbf{x})$ is probabilistic and can output well-calibrated $\Pr(y|\mathbf{x})$ values.
2. Provide interactive speed to user in loop even when D is very large
   - even a single sequential scan on D may not be practical.
   - D can be accessed only via an index.
3. Require user to label as few additional instances as possible
   - Similar to active learning but different ...
     - Active learning usually used in the context of learning classifiers
     - Task of learning classifiers different from evaluating given classifier

Our goal

1. Handle arbitrary classifier e.g. a user script
   - existing work assumes that $C(\mathbf{x})$ is probabilistic and can output well-calibrated $\Pr(y|\mathbf{x})$ values.
2. Provide interactive speed to user in loop even when D is very large
   - even a single sequential scan on D may not be practical.
   - D can be accessed only via an index.
3. Require user to label as few additional instances as possible
   - Similar to active learning but different ...
     - Active learning usually used in the context of learning classifiers
     - Task of learning classifiers different from evaluating given classifier

Our goal

1. Handle arbitrary classifier e.g. a user script
   - existing work assumes that $C(\mathbf{x})$ is probabilistic and can output well-calibrated $\Pr(y|\mathbf{x})$ values.
2. Provide interactive speed to user in loop even when D is very large
   - even a single sequential scan on D may not be practical.
   - D can be accessed only via an index.
3. Require user to label as few additional instances as possible
   - Similar to active learning but different ...
     - Active learning usually used in the context of learning classifiers
     - Task of learning classifiers different from evaluating given classifier

Our goal

1. Handle arbitrary classifier e.g. a user script
   - existing work assumes that $C(\mathbf{x})$ is probabilistic and can output well-calibrated $\Pr(y|\mathbf{x})$ values.
2. Provide interactive speed to user in loop even when D is very large
   - even a single sequential scan on D may not be practical.
   - D can be accessed only via an index.
3. Require user to label as few additional instances as possible
   - Similar to active learning but different ...
     - Active learning usually used in the context of learning classifiers
     - Task of learning classifiers different from evaluating given classifier

Outline

The problem has two aspects

1. **Accuracy estimation** (What this talk is about)
   - Given a fixed L what is the best estimator of μ?
   - How to do this scalably on large D?
2. **Instance selection** (Not covered in this talk, details in paper)
   - Selecting instances from D to be labeled by a human and adding to L. Performed in a loop.

Accuracy estimation

- *Simple averaged estimate* : $\hat{\mu}_R = \frac{1}{n} \sum_{i \in L} a_i$ is poor when L is small or biased
- A classical solution : stratified estimate
  - Stratify L and D into B buckets $(L_1, D_1), \ldots, (L_B, D_B)$
  - Measure accuracy $\hat{\mu}_b$ of L_b in each bucket b
  - Estimate weight w_b as fraction of D instances in bucket b$= \frac{|D_b|}{|D|}$
  - Stratified estimate $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b$

  Error of $\hat{\mu}_S <<$ Error of $\hat{\mu}_R$ if instances within a bucket are homogeneous

Two challenges

1. Selecting a stratification strategy that puts instances with same error in the same bucket
2. Finding w_b scalably when D is large $\Rightarrow$ cannot stratify whole of D.

Accuracy estimation

- *Simple averaged estimate* : $\hat{\mu}_R = \frac{1}{n}\sum_{i \in L} a_i$ is poor when L is small or biased
- A classical solution : stratified estimate
  - Stratify L and D into B buckets $(L_1, D_1), \ldots, (L_B, D_B)$
  - Measure accuracy $\hat{\mu}_b$ of L_b in each bucket b
  - Estimate weight w_b as fraction of D instances in bucket b$= \frac{|D_b|}{|D|}$
  - Stratified estimate $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b$

  Error of $\hat{\mu}_S <<$ Error of $\hat{\mu}_R$ if instances within a bucket are homogeneous

Two challenges

1. Selecting a stratification strategy that puts instances with same error in the same bucket
2. Finding w_b scalably when D is large $\Rightarrow$ cannot stratify whole of D.

Accuracy estimation

- *Simple averaged estimate* : $\hat{\mu}_R = \frac{1}{n}\sum_{i \in L} a_i$ is poor when L is small or biased
- A classical solution : stratified estimate
  - Stratify L and D into B buckets $(L_1, D_1), \ldots, (L_B, D_B)$
  - Measure accuracy $\hat{\mu}_b$ of L_b in each bucket b
  - Estimate weight w_b as fraction of D instances in bucket b= $\frac{|D_b|}{|D|}$
  - Stratified estimate $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b$

  Error of $\hat{\mu}_S <<$ Error of $\hat{\mu}_R$ if instances within a bucket are homogeneous

Two challenges

1. Selecting a stratification strategy that puts instances with same error in the same bucket
2. Finding w_b scalably when D is large $\Rightarrow$ cannot stratify whole of D.

Accuracy estimation

- *Simple averaged estimate* : $\hat{\mu}_R = \frac{1}{n}\sum_{i \in L} a_i$ is poor when L is small or biased
- A classical solution : stratified estimate
  - Stratify L and D into B buckets $(L_1, D_1), \ldots, (L_B, D_B)$
  - Measure accuracy $\hat{\mu}_b$ of L_b in each bucket b
  - Estimate weight w_b as fraction of D instances in bucket b$= \frac{|D_b|}{|D|}$
  - Stratified estimate $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b$

  Error of $\hat{\mu}_S <<$ Error of $\hat{\mu}_R$ if instances within a bucket are homogeneous

Two challenges

1. Selecting a stratification strategy that puts instances with same error in the same bucket
2. Finding w_b scalably when D is large $\Rightarrow$ cannot stratify whole of D.

Accuracy estimation

- *Simple averaged estimate* : $\hat{\mu}_R = \frac{1}{n}\sum_{i \in L} a_i$ is poor when L is small or biased
- A classical solution : stratified estimate
  - Stratify L and D into B buckets $(L_1, D_1), \ldots, (L_B, D_B)$
  - Measure accuracy $\hat{\mu}_b$ of L_b in each bucket b
  - Estimate weight w_b as fraction of D instances in bucket b$= \frac{|D_b|}{|D|}$
  - Stratified estimate $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b$

  Error of $\hat{\mu}_S <<$ Error of $\hat{\mu}_R$ if instances within a bucket are homogeneous

Two challenges

1. Selecting a stratification strategy that puts instances with same error in the same bucket
2. Finding w_b scalably when D is large $\Rightarrow$ cannot stratify whole of D.

Stratification strategy (h)

- **Existing approach** : Bin $\Pr(y|\mathbf{x})$ values assumed to be provided by a classifier
  - Bennett and Carvalho & Druck and McCallum
  - Not applicable since we wish to handle arbitrary classifiers.
- **Our Proposal** :
  - $F(\mathbf{x}, C)$: A feature representation of the input $\mathbf{x}$ and the result of deploying C on $\mathbf{x}$
  - Learn a hash function h on features using the labeled data L
  - Stratification evolves as more labeled instances get added to L (fixed in existing approaches)

Stratification strategy (h)

- **Existing approach** : Bin $\Pr(y|\mathbf{x})$ values assumed to be provided by a classifier
  - Bennett and Carvalho & Druck and McCallum
  - Not applicable since we wish to handle arbitrary classifiers.
- **Our Proposal** :
  - $F(\mathbf{x}, C)$: A feature representation of the input $\mathbf{x}$ and the result of deploying C on $\mathbf{x}$
  - Learn a hash function h on features using the labeled data L
  - Stratification evolves as more labeled instances get added to L (fixed in existing approaches)

Stratification strategy (h)

- **Existing approach** : Bin $\Pr(y|\mathbf{x})$ values assumed to be provided by a classifier
  - Bennett and Carvalho & Druck and McCallum
  - Not applicable since we wish to handle arbitrary classifiers.
- **Our Proposal** :
  - $F(\mathbf{x}, C)$: A feature representation of the input $\mathbf{x}$ and the result of deploying C on $\mathbf{x}$
  - Learn a hash function h on features using the labeled data L
  - Stratification evolves as more labeled instances get added to L (fixed in existing approaches)

Stratification strategy (h)

- **Existing approach** : Bin $\Pr(y|\mathbf{x})$ values assumed to be provided by a classifier
  - Bennett and Carvalho & Druck and McCallum
  - Not applicable since we wish to handle arbitrary classifiers.
- **Our Proposal** :
  - $F(\mathbf{x}, C)$: A feature representation of the input $\mathbf{x}$ and the result of deploying C on $\mathbf{x}$
  - Learn a hash function h on features using the labeled data L
  - Stratification evolves as more labeled instances get added to L (fixed in existing approaches)

Learning hyperplanes

Background

- *Hash function* : concatenation of r bits
  - bit $k = \text{sign}(\mathbf{w}_k.\mathbf{F}(x))$ i.e. the side of $\mathbf{w}_k$ on which x lies
  - $\mathbf{w}_k$: parameters to be learnt
- Learning problem : non-smooth and non-convex
- Existing work : learning distance-preserving hash function
  - Learn $\mathbf{w}_k$ parameters sequentially & smooth the sign function through various tricks
- Our problem can be expressed as special case of learning distance-preserving hash function
  - Exploit 0/1 nature of accuracy values rather than optimizing a black-box distance measure
  - Allows a more efficient algorithm

Learning hyperplanes

Background

- *Hash function* : concatenation of r bits
  - bit $k = \text{sign}(\mathbf{w}_k.\mathbf{F}(x))$ i.e. the side of $\mathbf{w}_k$ on which x lies
  - $\mathbf{w}_k$: parameters to be learnt
- Learning problem : non-smooth and non-convex
- Existing work : learning distance-preserving hash function
  - Learn $\mathbf{w}_k$ parameters sequentially & smooth the sign function through various tricks
- Our problem can be expressed as special case of learning distance-preserving hash function
  - Exploit 0/1 nature of accuracy values rather than optimizing a black-box distance measure
  - Allows a more efficient algorithm

Learning hyperplanes

Background

- *Hash function* : concatenation of r bits
  - bit $k = \text{sign}(\mathbf{w}_k.\mathbf{F}(x))$ i.e. the side of $\mathbf{w}_k$ on which x lies
  - $\mathbf{w}_k$: parameters to be learnt
- Learning problem : non-smooth and non-convex
- Existing work : learning distance-preserving hash function
  - Learn $\mathbf{w}_k$ parameters sequentially & smooth the sign function through various tricks
- Our problem can be expressed as special case of learning distance-preserving hash function
  - Exploit 0/1 nature of accuracy values rather than optimizing a black-box distance measure
  - Allows a more efficient algorithm

Learning hyperplanes

Background

- *Hash function* : concatenation of r bits
  - bit $k = \text{sign}(\mathbf{w}_k.\mathbf{F}(x))$ i.e. the side of $\mathbf{w}_k$ on which x lies
  - $\mathbf{w}_k$: parameters to be learnt
- Learning problem : non-smooth and non-convex
- Existing work : learning distance-preserving hash function
  - Learn $\mathbf{w}_k$ parameters sequentially & smooth the sign function through various tricks
- Our problem can be expressed as special case of learning distance-preserving hash function
  - Exploit 0/1 nature of accuracy values rather than optimizing a black-box distance measure
  - Allows a more efficient algorithm

Learning hyperplanes

Background

- *Hash function* : concatenation of r bits
  - bit $k = \text{sign}(\mathbf{w}_k.\mathbf{F}(x))$ i.e. the side of $\mathbf{w}_k$ on which x lies
  - $\mathbf{w}_k$: parameters to be learnt
- Learning problem : non-smooth and non-convex
- Existing work : learning distance-preserving hash function
  - Learn $\mathbf{w}_k$ parameters sequentially & smooth the sign function through various tricks
- Our problem can be expressed as special case of learning distance-preserving hash function
  - Exploit 0/1 nature of accuracy values rather than optimizing a black-box distance measure
  - Allows a more efficient algorithm

Learning hyperplanes

Our approach

- *Main step* : find the best value of $\mathbf{w}_k$ for a bit k, assuming hyperplanes of other bits are fixed
- For each bucket formed from remaining hyperplanes, arbitrarily choose which side of hyperplane $\mathbf{w}_k$ it wants to call +ve or -ve
- Use logistic loss to find the optimal $\mathbf{w}_k$ so that in each bucket, $\mathbf{w}_k$ correctly puts points on the "right" side
    - Can be solved optimally using a standard logistic classifier
- Use a EM-like iteration to refine the side chosen as positive in each bucket until convergence

We see in our experiments that our method provides much better results than existing algorithms

Learning hyperplanes

Our approach

- *Main step* : find the best value of $\mathbf{w}_k$ for a bit k, assuming hyperplanes of other bits are fixed
- For each bucket formed from remaining hyperplanes, arbitrarily choose which side of hyperplane $\mathbf{w}_k$ it wants to call +ve or -ve
- Use logistic loss to find the optimal $\mathbf{w}_k$ so that in each bucket, $\mathbf{w}_k$ correctly puts points on the "right" side
  - Can be solved optimally using a standard logistic classifier
- Use a EM-like iteration to refine the side chosen as positive in each bucket until convergence

We see in our experiments that our method provides much better results than existing algorithms

Learning hyperplanes

Our approach

- *Main step* : find the best value of $\mathbf{w}_k$ for a bit k, assuming hyperplanes of other bits are fixed
- For each bucket formed from remaining hyperplanes, arbitrarily choose which side of hyperplane $\mathbf{w}_k$ it wants to call +ve or -ve
- Use logistic loss to find the optimal $\mathbf{w}_k$ so that in each bucket, $\mathbf{w}_k$ correctly puts points on the "right" side
    - Can be solved optimally using a standard logistic classifier
- Use a EM-like iteration to refine the side chosen as positive in each bucket until convergence

We see in our experiments that our method provides much better results than existing algorithms

Learning hyperplanes

Our approach

- *Main step* : find the best value of $\mathbf{w}_k$ for a bit k, assuming hyperplanes of other bits are fixed
- For each bucket formed from remaining hyperplanes, arbitrarily choose which side of hyperplane $\mathbf{w}_k$ it wants to call +ve or -ve
- Use logistic loss to find the optimal $\mathbf{w}_k$ so that in each bucket, $\mathbf{w}_k$ correctly puts points on the "right" side
  - Can be solved optimally using a standard logistic classifier
- Use a EM-like iteration to refine the side chosen as positive in each bucket until convergence

We see in our experiments that our method provides much better results than existing algorithms

Learning hyperplanes

Our approach

- *Main step* : find the best value of $\mathbf{w}_k$ for a bit k, assuming hyperplanes of other bits are fixed
- For each bucket formed from remaining hyperplanes, arbitrarily choose which side of hyperplane $\mathbf{w}_k$ it wants to call +ve or -ve
- Use logistic loss to find the optimal $\mathbf{w}_k$ so that in each bucket, $\mathbf{w}_k$ correctly puts points on the "right" side
  - Can be solved optimally using a standard logistic classifier
- Use a EM-like iteration to refine the side chosen as positive in each bucket until convergence

We see in our experiments that our method provides much better results than existing algorithms

Learning hyperplanes

Our approach

- *Main step* : find the best value of $\mathbf{w}_k$ for a bit k, assuming hyperplanes of other bits are fixed
- For each bucket formed from remaining hyperplanes, arbitrarily choose which side of hyperplane $\mathbf{w}_k$ it wants to call +ve or -ve
- Use logistic loss to find the optimal $\mathbf{w}_k$ so that in each bucket, $\mathbf{w}_k$ correctly puts points on the "right" side
  - Can be solved optimally using a standard logistic classifier
- Use a EM-like iteration to refine the side chosen as positive in each bucket until convergence

We see in our experiments that our method provides much better results than existing algorithms

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Estimating bucket weights without sequentially hashing D

- $\hat{\mu}_S = \sum_b w_b \hat{\mu}_b = \frac{1}{N} \sum_{i \in D} \hat{\mu}_{h(\mathbf{f}_i)}$
- Proposal sampling : $\hat{\mu}_{S_q} = \frac{1}{N} \left(\frac{1}{m} \sum_{i \in Q} \frac{\hat{\mu}_{h(\mathbf{f}_i)}}{q(i)} \right)$
- Optimal $q(i) \propto \hat{\mu}_{h(\mathbf{f}_i)}$: impossible without assigning each $i \in D$ to a bucket of $h(.)$
- Allowed $q(i)$ are the ones which assign same probability to all instances i within an index partition D_u of D
- **Claim** : Under above restriction, $q_u \propto \sqrt{\sum_b \hat{\mu}_b^2 p(b|u)}$ where $p(b|u)$ is the fraction of $i \in D_u$ with $h(\mathbf{f}_i) = b$
- $p(b|u)$ estimate : Initially depend on labeled data & small static sample. Refine estimate as more instances get sampled from D_u

Results

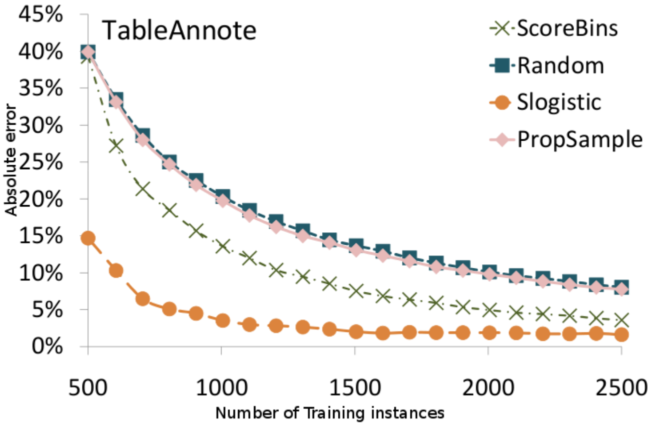
Summary of datasets used

- **TableAnnote** : Annotate columns of Web tables to type nodes of an ontology
- **Spam** : Classifying web-pages as spam or not
- **DNA** : Binary DNA classification task
- **HomeGround, HomePredicted** : Dataset of (entity, web-page) instances and decide if web-page was a homepage for the entity

Dataset	#	Size		Accuracy (%)	
	Features	Seed(L)	Unlabeled(D)	Seed(L)	True(D)
TableAnnote	42	541	11,954,983	56.4	16.5
Spam	1000	5000	350,000	86.4	93.2
DNA	800	100,000	50,000,000	72.2	77.9
HomeGround	66	514	1060	50.4	32.8
HomePredicted	66	8658	13,951,053	83.2	93.9

Results

Comparison of estimation strategies on the TableAnnote dataset



Random	Random Sampling
PropSample	Sampling from proposal distribution (Sawade et al., 2010)
ScoreBins	Stratified sampling with scores

Results

Comparison of estimation strategies on remaining datasets

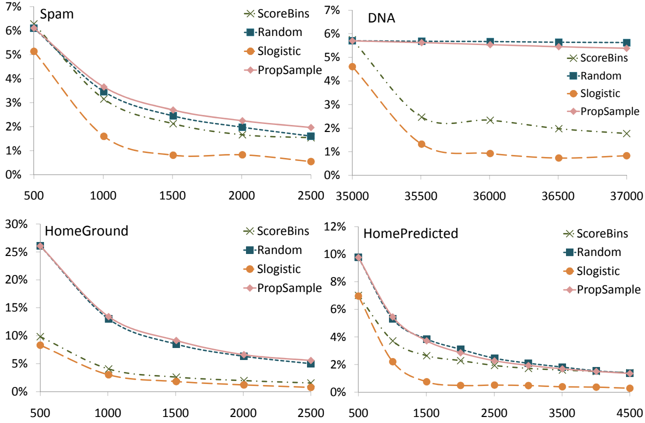
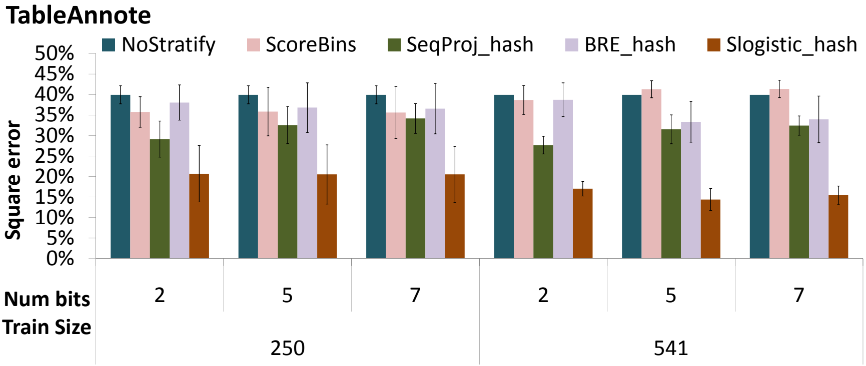


Figure: Absolute error (on the Y axis) of different estimation algorithms against increasing number of labeled instances (on the X axis)

Results

Comparison of different stratification methods on the TableAnnote dataset



NoStratify	Simple averaging (no stratification)
ScoreBins	Stratify using classifier scores
SeqProj_hash	Learn hyperplanes using method in Wang et al.
BRE_hash	Learn hyperplanes using method in Kulis and Darrell

Results

Comparison of different stratification methods on remaining datasets

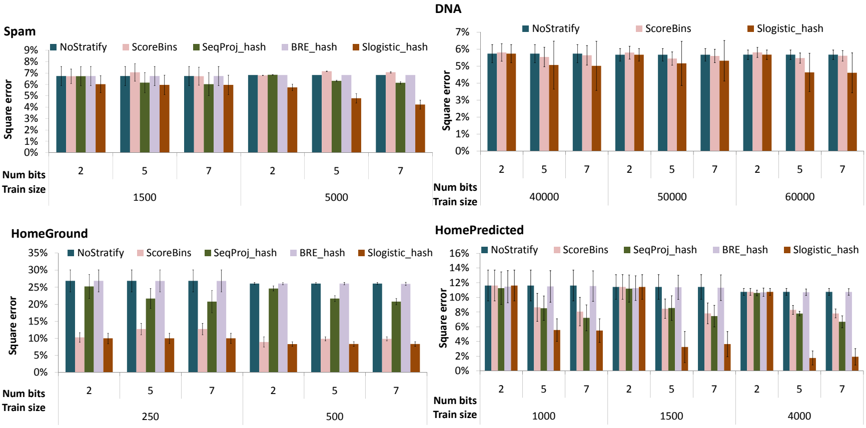


Figure: Error of different stratification methods against increasing training sizes and for different number of bits

Results

Comparison of methods of sampling from indexed data for estimating bucket weights

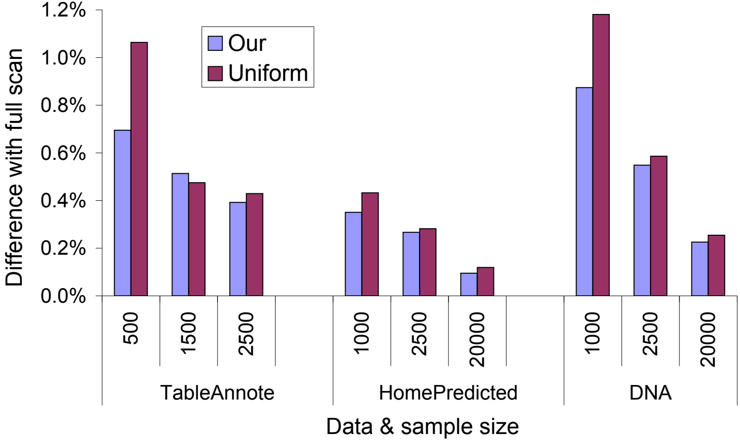


Figure: Comparing methods of sampling from indexed data for estimating bucket weights

Summary

1. Addressed the challenge of *calibrating a classifier's accuracy on large unlabeled datasets given small amounts of labeled data and a human labeler*
2. Proposed a stratified sampling-based method for accuracy estimation that provides better estimates than simple averaging & better selection of instances for labeling than random sampling
3. Between 15% and 62% relative reduction in error achieved compared to existing approaches
4. Algorithm made *scalable* by proposing optimal sampling strategies for accessing indexed unlabeled data directly
5. Close to optimal performance while reading three orders of magnitude fewer instances on large datasets

Summary

1. Addressed the challenge of *calibrating a classifier's accuracy on large unlabeled datasets given small amounts of labeled data and a human labeler*
2. Proposed a stratified sampling-based method for accuracy estimation that provides better estimates than simple averaging & better selection of instances for labeling than random sampling
3. Between 15% and 62% relative reduction in error achieved compared to existing approaches
4. Algorithm made *scalable* by proposing optimal sampling strategies for accessing indexed unlabeled data directly
5. Close to optimal performance while reading three orders of magnitude fewer instances on large datasets

Summary

1. Addressed the challenge of *calibrating a classifier's accuracy on large unlabeled datasets given small amounts of labeled data and a human labeler*
2. Proposed a stratified sampling-based method for accuracy estimation that provides better estimates than simple averaging & better selection of instances for labeling than random sampling
3. Between 15% and 62% relative reduction in error achieved compared to existing approaches
4. Algorithm made *scalable* by proposing optimal sampling strategies for accessing indexed unlabeled data directly
5. Close to optimal performance while reading three orders of magnitude fewer instances on large datasets

Summary

1. Addressed the challenge of *calibrating a classifier's accuracy on large unlabeled datasets given small amounts of labeled data and a human labeler*
2. Proposed a stratified sampling-based method for accuracy estimation that provides better estimates than simple averaging & better selection of instances for labeling than random sampling
3. Between 15% and 62% relative reduction in error achieved compared to existing approaches
4. Algorithm made *scalable* by proposing optimal sampling strategies for accessing indexed unlabeled data directly
5. Close to optimal performance while reading three orders of magnitude fewer instances on large datasets

Summary

1. Addressed the challenge of *calibrating a classifier's accuracy on large unlabeled datasets given small amounts of labeled data and a human labeler*
2. Proposed a stratified sampling-based method for accuracy estimation that provides better estimates than simple averaging & better selection of instances for labeling than random sampling
3. Between 15% and 62% relative reduction in error achieved compared to existing approaches
4. Algorithm made *scalable* by proposing optimal sampling strategies for accessing indexed unlabeled data directly
5. Close to optimal performance while reading three orders of magnitude fewer instances on large datasets

Thank You

References I

Bennett, P. N. and Carvalho, V. R. (2010). Online stratified sampling: evaluating classifiers at web-scale. In *CIKM*.

Druck, G. and McCallum, A. (2011). Toward interactive training and evaluation. In *CIKM*.

Kulis, B. and Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In *NIPS*.

Sawade, C., Landwehr, N., Bickel, S., and Scheffer, T. (2010). Active risk estimation. In *ICML*.

Wang, J., Kumar, S., and Chang, S. (2010). Sequential projection learning for hashing with compact codes. In *ICML*.