Association Rules and Clustering

Steve Avsec

Illinois Institute of Technology

April 15, 2024

Overview

1. One Last DL Note
2. Association Rules
3. Min Hashing

One Last Note on Convexity

Just as a point of clarification:

- Non-convexity of the loss function in DL is not a ”bad” thing.

One Last Note on Convexity

Just as a point of clarification:

- Non-convexity of the loss function in DL is not a ”bad” thing.
- Convex functions are a small, special class of functions that are easy to compute with.

One Last Note on Convexity

Just as a point of clarification:

- Non-convexity of the loss function in DL is not a ”bad” thing.
- Convex functions are a small, special class of functions that are easy to compute with.
- Many interesting physical systems are studied using non-convex optimization problems.

One Last Note on Convexity

Just as a point of clarification:

- Non-convexity of the loss function in DL is not a ”bad” thing.
- Convex functions are a small, special class of functions that are easy to compute with.
- Many interesting physical systems are studied using non-convex optimization problems.
- The class of non-convex optimization functions is NP-complete.

The Setup

- We have a set of ”items” $\{i_1, \ldots, i_L\}$.

The Setup

- We have a set of ”items” $\{i_1, \ldots, i_L\}$.
- We have database of ”transactions” $\{X_1, \ldots, X_N\}$.

The Setup

- We have a set of "items" $\{i_1, \ldots, i_L\}$.
- We have database of "transactions" $\{X_1, \ldots, X_N\}$.
- A "typical" setup is that each X_j consists of a unique id plus a k-dimensional vector describinge each item in the transaction.

The Setup

- We have a set of ”items” $\{i_1, \dots, i_L\}$.
- We have database of ”transactions” $\{X_1, \dots, X_N\}$.
- A ”typical” setup is that each X_j consists of a unique id plus a k-dimensional vector describinge each item in the transaction.
- The basic goal is to estimate $P(v)$ where v is a particular feature vector.

Item Sets

- Let S_l denote the set of values each feature vector can take.

Item Sets

- Let S_l denote the set of values each feature vector can take.
- Let

$$K = \sum_{l=1}^{k} |S_l|$$

Item Sets

- Let S_l denote the set of values each feature vector can take.
- Let

$$K = \sum_{l=1}^{k} |S_l|$$

- Let Z_k denote a dummy variable $\{0, 1\}$-valued that indicates if the corresponding item takes a particular value.

Item Sets

- Let S_l denote the set of values each feature vector can take.
- Let
$$K = \sum_{l=1}^{k} |S_l|$$
- Let Z_k denote a dummy variable $\{0, 1\}$-valued that indicates if the corresponding item takes a particular value.
- Goal: Find subsets $\mathcal{K} \subset \{1, \ldots, K\}$ such that
$$Pr\left(\cap_{k \in \mathcal{K}} Z_k = 1\right) = Pr\left(\prod_{k \in \mathcal{K}} Z_k = 1\right)$$
is "large".

Prevalence

Consider the empirical probability

$$\hat{Pr}\left(\prod_{k\in\mathcal{K}} Z_k = 1\right) = \frac{1}{N}\sum_{j=1}^{N}\prod_{k\in\mathcal{K}} z_{j,k}$$

where $z_{j,k}$ is the value of Z_k for the jth transaction.

Prevalence

Consider the empirical probability

$$\hat{Pr}\left(\prod_{k\in\mathcal{K}} Z_k = 1\right) = \frac{1}{N}\sum_{j=1}^{N}\prod_{k\in\mathcal{K}} z_{j,k}$$

where $z_{j,k}$ is the value of Z_k for the jth transaction.

This quantity is called the "prevalence" of $\mathcal{K}$ and is denoted $\mathcal{T}(\mathcal{K})$.

Prevalence

Consider the empirical probability

$$\hat{Pr}\left(\prod_{k\in\mathcal{K}} Z_k = 1\right) = \frac{1}{N}\sum_{j=1}^{N}\prod_{k\in\mathcal{K}} z_{j,k}$$

where $z_{j,k}$ is the value of Z_k for the jth transaction.

This quantity is called the "prevalence" of $\mathcal{K}$ and is denoted $\mathcal{T}(\mathcal{K})$.

We will look for all item sets such that

$$\{\mathcal{K}|\mathcal{T}(\mathcal{K}) > t\}$$

for some fixed threshold t.

The A Priori Algorithm

In ”typical”, real world data,

1. The set
$$\{\mathcal{K} | \mathcal{T}(\mathcal{K}) > t\}$$
is relatively small.
2. If $\mathcal{L} \subseteq \mathcal{K}$ then $\mathcal{T}(\mathcal{L}) \leq \mathcal{T}(\mathcal{K})$.

The A Priori Algorithm

In "typical", real world data,

1. The set
$$\{\mathcal{K} | \mathcal{T}(\mathcal{K}) > t\}$$
is relatively small.
2. If $\mathcal{L} \subseteq \mathcal{K}$ then $\mathcal{T}(\mathcal{L}) \leq \mathcal{T}(\mathcal{K})$.

First, compute the prevalence for all single-item sets and keep any items where the prevalence is greater than t.

The A Priori Algorithm

In "typical", real world data,

1. The set
$$\{\mathcal{K} | \mathcal{T}(\mathcal{K}) > t\}$$
is relatively small.
2. If $\mathcal{L} \subseteq \mathcal{K}$ then $\mathcal{T}(\mathcal{L}) \leq \mathcal{T}(\mathcal{K})$.

First, compute the prevalence for all single-item sets and keep any items where the prevalence is greater than t.

Then compute the prevalence for all two-item sets where each item passed the previous iteration.

The A Priori Algorithm

In "typical", real world data,

1. The set
$$\{\mathcal{K} | \mathcal{T}(\mathcal{K}) > t\}$$
is relatively small.
2. If $\mathcal{L} \subseteq \mathcal{K}$ then $\mathcal{T}(\mathcal{L}) \leq \mathcal{T}(\mathcal{K})$.

First, compute the prevalence for all single-item sets and keep any items where the prevalence is greater than t.

Then compute the prevalence for all two-item sets where each item passed the previous iteration.

Continue until all sets have prevalence less than the threshold.

Association Rules

For a specific subset $\mathcal{K}$ such that $\mathcal{T}(\mathcal{K}) > t$, an *association rule* is just a splitting

$$A \cup B = \mathcal{K} \text{ and } A \cap B = \emptyset$$

Association Rules

For a specific subset $\mathcal{K}$ such that $\mathcal{T}(\mathcal{K}) > t$, an *association rule* is just a splitting

$$A \cup B = \mathcal{K} \text{ and } A \cap B = \emptyset$$

In this case, we write $A \Rightarrow B$, and denote $T(A \Rightarrow B) = \mathcal{T}(\mathcal{K})$.

Association Rules

For a specific subset $\mathcal{K}$ such that $\mathcal{T}(\mathcal{K}) > t$, an *association rule* is just a splitting

$$A \cup B = \mathcal{K} \text{ and } A \cap B = \emptyset$$

In this case, we write $A \Rightarrow B$, and denote $T(A \Rightarrow B) = \mathcal{T}(\mathcal{K})$. We define the *confidence* $C(A \Rightarrow B)$ of the rule by

$$C(A \Rightarrow B) = \frac{T(A \Rightarrow B)}{T(A)}$$

which is just an estimate of $Pr(B|A)$.

Association Rules

For a specific subset $\mathcal{K}$ such that $\mathcal{T}(\mathcal{K}) > t$, an *association rule* is just a splitting

$$A \cup B = \mathcal{K} \text{ and } A \cap B = \emptyset$$

In this case, we write $A \Rightarrow B$, and denote $T(A \Rightarrow B) = \mathcal{T}(\mathcal{K})$. We define the *confidence* $C(A \Rightarrow B)$ of the rule by

$$C(A \Rightarrow B) = \frac{T(A \Rightarrow B)}{T(A)}$$

which is just an estimate of $Pr(B|A)$.
Define the *lift* $L(A \Rightarrow B)$ by

$$L(A \Rightarrow B) = \frac{C(A \Rightarrow B)}{T(B)}$$

Output

We want to find association rules $A \Rightarrow B$ such that

1. $T(A \Rightarrow B) > t$
2. $C(A \Rightarrow B) > c$

Output

We want to find association rules $A \Rightarrow B$ such that

1. $T(A \Rightarrow B) > t$
2. $C(A \Rightarrow B) > c$

The original A Priori algorithm paper (Agrawal et. al. 1995) provides a technique for ”pruning” rules for a given subset $\mathcal{K}$ so one doesn’t have to just check all of the subsets to optimize $C(A \Rightarrow B)$.

Output

We want to find association rules $A \Rightarrow B$ such that

1. $T(A \Rightarrow B) > t$
2. $C(A \Rightarrow B) > c$

The original A Priori algorithm paper (Agrawal et. al. 1995) provides a technique for "pruning" rules for a given subset $\mathcal{K}$ so one doesn't have to just check all of the subsets to optimize $C(A \Rightarrow B)$.

In the end, we end up with a list of rules $A \Rightarrow B$ such that the two conditions above are met.

Pros and Cons

- Pro: Simplicity and interpretability

Pros and Cons

- Pro: Simplicity and interpretability
- Pro: Can be run entirely in-database, highly scalable.

Pros and Cons

- Pro: Simplicity and interpretability
- Pro: Can be run entirely in-database, highly scalable.
- Con: Items with small prevalence are dropped early and not considered.

Pros and Cons

- Pro: Simplicity and interpretability
- Pro: Can be run entirely in-database, highly scalable.
- Con: Items with small prevalence are dropped early and not considered.
- Con: Will miss strong signals with infrequent data.

The Setup

This section is taken from Chapter 3the book *Mining Massive Datasets* by Leskovec, Rajaraman, and Ullman.

The Setup

This section is taken from Chapter 3the book *Mining Massive Datasets* by Leskovec, Rajaraman, and Ullman.

The basic idea is to find similar documents in a corpus given an input document.

The Setup

This section is taken from Chapter 3the book *Mining Massive Datasets* by Leskovec, Rajaraman, and Ullman.

The basic idea is to find similar documents in a corpus given an input document.

Jaccard similarity: Given finite sets A and B,

$$\mathrm{Sim}_J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

The Setup

This section is taken from Chapter 3the book *Mining Massive Datasets* by Leskovec, Rajaraman, and Ullman.

The basic idea is to find similar documents in a corpus given an input document.

Jaccard similarity: Given finite sets A and B,

$$\mathrm{Sim}_J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Sometimes we also see this expressed as a "distance":

$$d_J(A, B) = 1 - \mathrm{Sim}_J(A, B)$$

Shingling

A k-shingle of a document is any (consecutive) substring of the document.

Shingling

A *k*-shingle of a document is any (consecutive) substring of the document.

Example: If the document in question is "*abcde*", the 2-shingles are $\{"ab", "bc", "cd", "de"\}$.

Shingling

A k-shingle of a document is any (consecutive) substring of the document.

Example: If the document in question is $''abcde''$, the 2-shingles are $\{''ab'', ''bc'', ''cd'', ''de''\}$.

Another example: If the document in question is $''abracadabra''$, the 2-singles are

$$\{''ab'', ''br'', ''ra'', ''ac'', ''ca'', ''da''\}$$

Shingling

A k-shingle of a document is any (consecutive) substring of the document.

Example: If the document in question is "*abcde*", the 2-shingles are {"*ab*", "*bc*", "*cd*", "*de*"}.

Another example: If the document in question is "*abracadabra*", the 2-singles are

$$\{"ab", "br", "ra", "ac", "ca", "da"\}$$

There is no sure, fast rule to how long shingles should be, but values between 5 and 9 are common depending on the typical size of the documents (emails vs. websites vs. research papers)

Hashing

Next we can hash shingles to an integer (often a 32-bit int suffices).

Hashing

Next we can hash shingles to an integer (often a 32-bit int suffices).

There are alternative methods to shingling utilizing word tokenization and similar techniques, but hashing still plays the pivotal role.

Characteristic Matrices

Let S be the universal set of all shingles and let $S_1, \ldots, S_N$ denote the sets of shingles for each document.

Characteristic Matrices

Let S be the universal set of all shingles and let $S_1, \ldots, S_N$ denote the sets of shingles for each document.

Let M denote the matrix such that

$$(m)_{ij} = \begin{cases} 1 & \text{if the } i\text{th shingle is in document } j \\ 0 & \text{otherwise} \end{cases}$$

Note here that the *columns* correspond to the samples in this case.

Characteristic Matrices

Let S be the universal set of all shingles and let $S_1, \ldots, S_N$ denote the sets of shingles for each document.

Let M denote the matrix such that

$$(m)_{ij} = \begin{cases} 1 & \text{if the } i\text{th shingle is in document } j \\ 0 & \text{otherwise} \end{cases}$$

Note here that the *columns* correspond to the samples in this case.

This matrix is called the *characteristic matrix*, and it is typically *very* sparse so it is typical to use a different data structure to represent the matrix.

Min hashing

- Permute the rows of M.

Min hashing

- Permute the rows of M.
- For each set S_j, find the first non-zero row.

Min hashing

- Permute the rows of M.
- For each set S_j, find the first non-zero row.
- Do this a few hundred times.

Min hashing

- Permute the rows of M.
- For each set S_j, find the first non-zero row.
- Do this a few hundred times.
- The number of min hashes that coincide for S_j and S_k turns out to be $\mathrm{sim}_J(S_j, S_k)$.