Etsy

ANDREW CLEGG @ BERLIN BUZZWORDS 2016

LEARNING TO RANK: WHERE SEARCH MEETS MACHINE LEARNING

1. Background & context
2. Feature engineering
3. Model training with SVMs
4. LTR in production

SECTION 1

Background & context

How does search relevance work?

“purple hand woven unicorn hair sweater”

hair ➡ 0.25
hand ➡ 0.09
purple ➡ 0.31
sweater ➡ 0.28
unicorn ➡ 0.69
woven ➡ 0.45

tfidf weights

Item	Query
hair ➡ 0.25	hair ➡ 0.00
hand ➡ 0.09	hand ➡ 0.00
purple ➡ 0.31	purple ➡ 2.35
sweater ➡ 0.28	sweater ➡ 1.98
unicorn ➡ 0.69	unicorn ➡ 0.00
woven ➡ 0.45	woven ➡ 0.00

idf weights

Item	cosine similarity	Query
hair ➡ 0.25		hair ➡ 0.00
hand ➡ 0.09		hand ➡ 0.00
purple ➡ 0.31		purple ➡ 2.35
sweater ➡ 0.28		sweater ➡ 1.98
unicorn ➡ 0.69		unicorn ➡ 0.00
woven ➡ 0.45		woven ➡ 0.00

Item

cosine similarity

Query

Item		Query
hair ➡ 0.25		hair ➡ 0.00
hand ➡ 0.09		hand ➡ 0.00
purple ➡ 0.31		purple ➡ 2.35
sweater ➡ 0.28		sweater ➡ 1.98
unicorn ➡ 0.69		unicorn ➡ 0.00
woven ➡ 0.45		woven ➡ 0.00

Warning: horrible over-simplification!

Documents with multiple fields

(title_score * 1.5) + (body_score * 1.0) + (comments_score * 0.25)

Non-textual boosts...

popularity: score * (num_clicks / num_impressions)

proximity to user: score / haversine_dist

age: score / (now - posting_date)

user favourited item: score * arbitrary_constant

... with some sensible scaling functions

popularity: f(score, num_clicks, num_impressions)

proximity to user: f(score, haversine_dist)

age: f(score, now, posting_date)

user favourited item: f(score, num_user_favourites)

But these functions must still contain scaling constants.

How can we combine all these factors?

f(title_score, body_score, comments_score, num_clicks, num_impressions, age, haversine_dist, num_user_favourites, user_favourited_this, …)

Some depend on item, some on query, some on user.

How to combine meaningfully?

How to keep “magic numbers” up-to-date?

TREAT IT AS A MACHINE LEARNING PROBLEM.

Represent each item as a vector of features

Each feature has a name and value.

TITLE_hair ➡ 0.25, USER_CLICKED_BEFORE ➡ 1

***Weighted sum* of feature values gives relevance score.**

But where do these weights come from?

Item

weighted sum

Model

Item	Model
TITLE_hair ➡ 0.25	TITLE_hair ➡ +0.01
TITLE_hand ➡ 0.09	TITLE_hand ➡ +0.03
TITLE_purple ➡ 0.31	TITLE_purple ➡ +0.14
TITLE_sweater ➡ 0.28	TITLE_sweater ➡ +0.08
USER_CLICKED ➡ 1.00	USER_CLICKED ➡ +0.46
AGE_YEARS ➡ 0.10	AGE_YEARS ➡ –0.12
	…

Build a target ranking from historical data

Query: “purple sweater”

ID62858 purple hand-woven unicorn-hair sweater
ID78923 colourless green sweater with furious purple ideas
ID19846 blue sweater decorated with purple figments
ID73956 purple yarn, ideal for making a sweater

Build a target ranking from historical data

Query: “purple sweater”

ID62858 ← most clicked = most relevant

ID78923 …

ID19846 …

ID73956 ← least clicked = least relevant

Trainer compares predicted ranking to target

Query: “purple sweater”

ID62858 ← predicted ranking is correct

ID19846

ID78923

¡FAIL?

ID73956 ← predicted ranking is correct

Tweak weights in direction that improves ranking

Query: “purple sweater”

ID62858
ID78923
ID19846
ID73956

Rinse and repeat, until ranking accuracy stops improving.

Warning: horrible over-simplification!

SECTION 2

Feature engineering

Representing items as features

TITLE_blue ➡ 0.24 (or 1.0)
DESC_suede ➡ 0.31 (or 1.0)
TAXO_shoes ➡ 0.16 (or 1.0)

CLICK_RATE ➡ 0.23
CONV_RATE ➡ 0.02
PRICE_QUANTILE ➡ 0.73

LDA_TOPIC_37 ➡ 0.67
LSI_TOPIC_12 ➡ 0.19

DOC_CLUSTER_3 ➡ 1.0
IMG_FEAT_17 ➡ 0.86

Example: "Images Don't Lie: Transferring Deep Visual Semantic Features to Large-Scale Multimodal Learning to Rank", Lynch et al., KDD 2016

How to include query context

A model with only item features learns a ‘global’ score.

Easy option: use as modifier for TFIDF relevance.

score = f(ltr_score, lucene_score)

But this takes a step backwards.

Modelling <query, item> pairs

TITLE_blue ➡ 0.24 (or 1.0)
DESC_suede ➡ 0.31 (or 1.0)
TAXO_shoes ➡ 0.16 (or 1.0)
CLICK_RATE ➡ 0.23
CONV_RATE ➡ 0.02
PRICE_QUANTILE ➡ 0.73

QPOS_nn_adj ➡ 1.0
QCAT_footwear ➡ 1.0
Q_ENTROPY ➡ 5.34

TFIDF_TITLE ➡ 7.86
BM25_DESC ➡ 8.96

Footnote: best to rescale features that aren't in 0-1 range if possible.

Explicit query-item interactions

QUERY_red:TITLE_scarlet ➡ 1.0

QUERY_red:TITLE_blue ➡ 1.0

Meaning: an item containing “scarlet” or “blue” appeared in results for a query containing “red”.

“QUERY” could also be “PAGE” or “CONTEXT” or even “USER”.

Query-specific ranking models

Or: train a *separate* model for each query in your logs.

(Or top-N most common queries.)

Generally works well, assuming plenty of data for each.

SECTION 3

Model training with SVMs

Imagine we're classifying spam emails

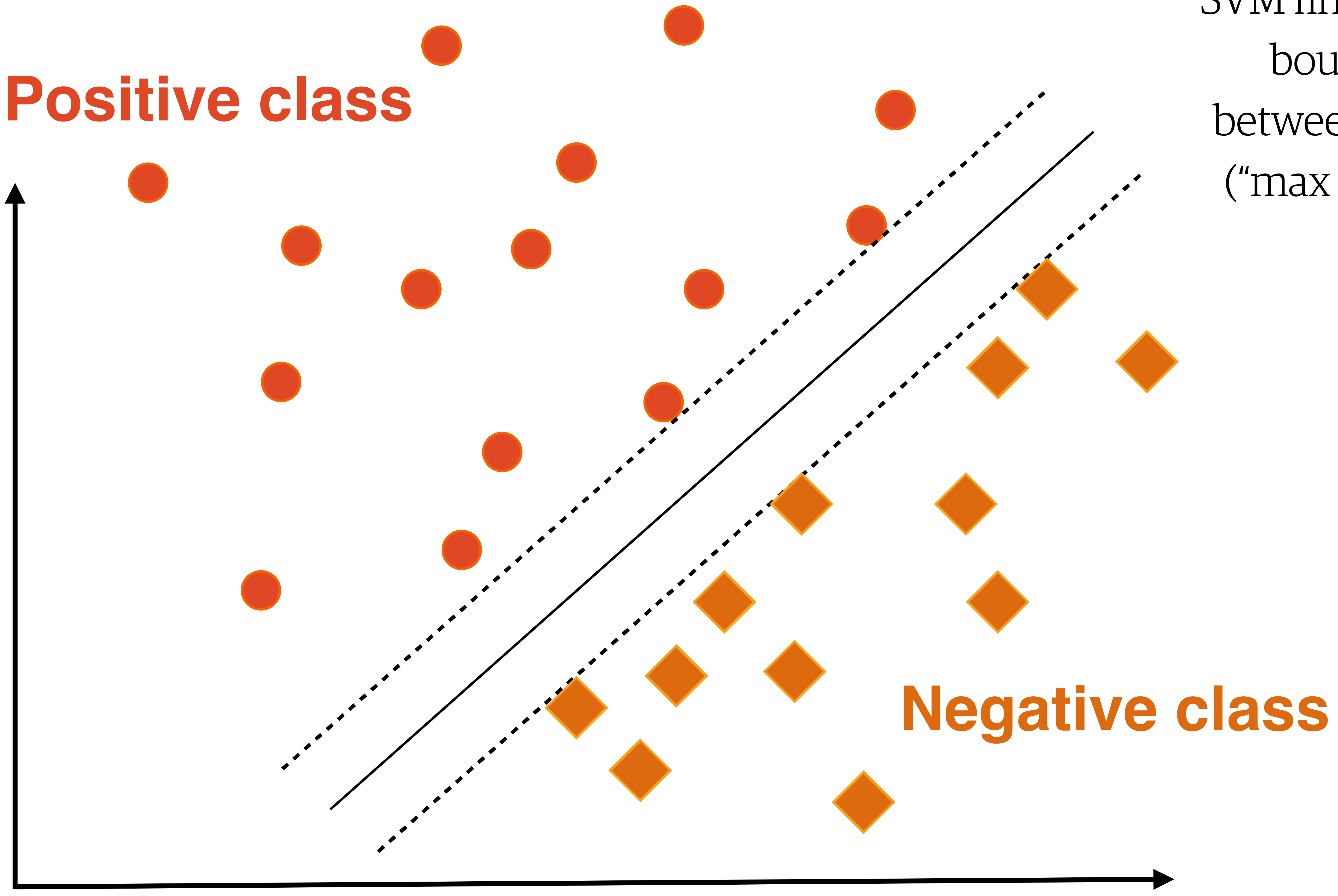
<email1> ↠ +1

… training instance 1 has been manually tagged as spam.

<email2> ↠ –1

… training instance 2 has been tagged as not-spam.

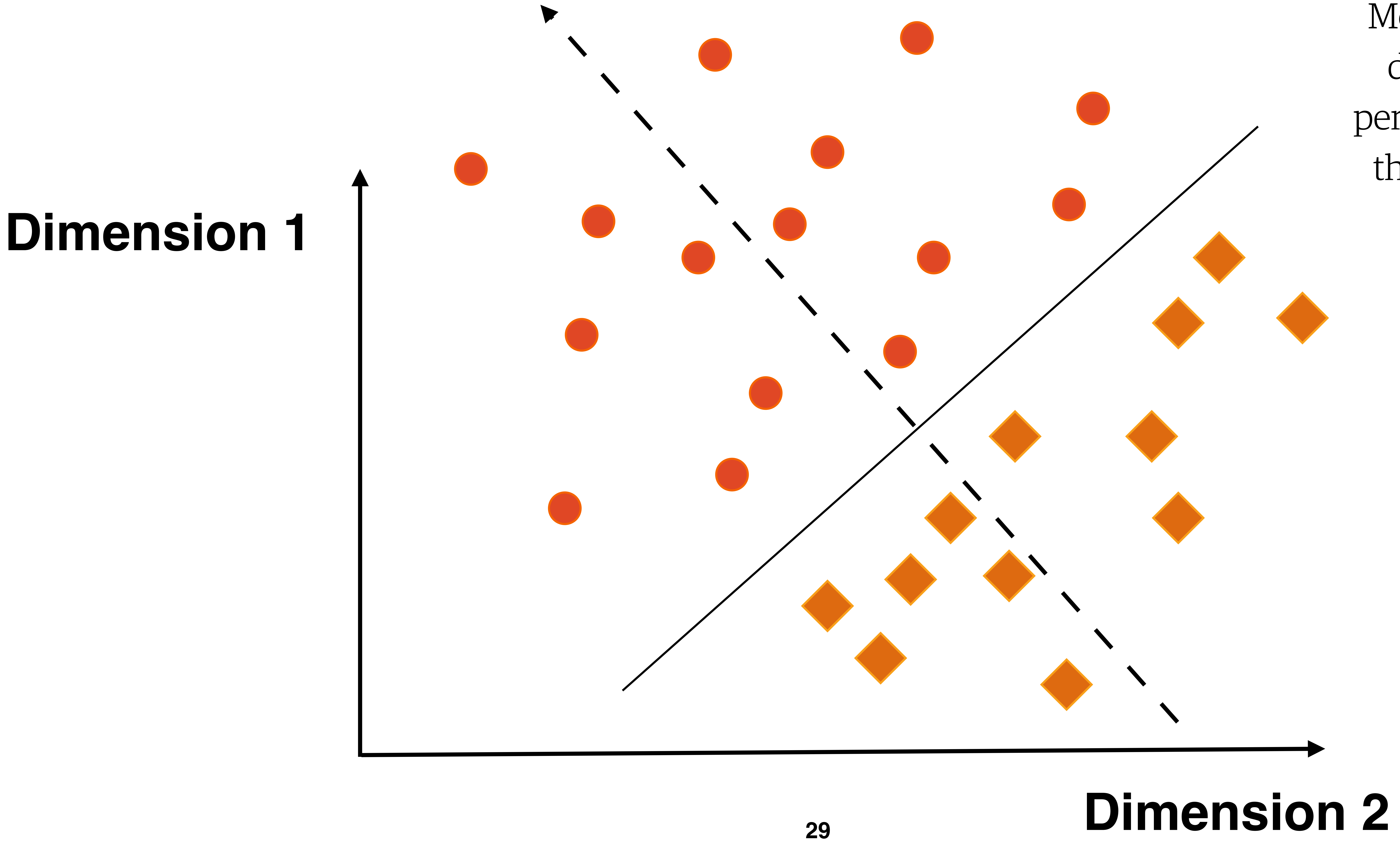
Support Vector Machines for classification



SVM finds best-fit boundary between classes (“max margin”)

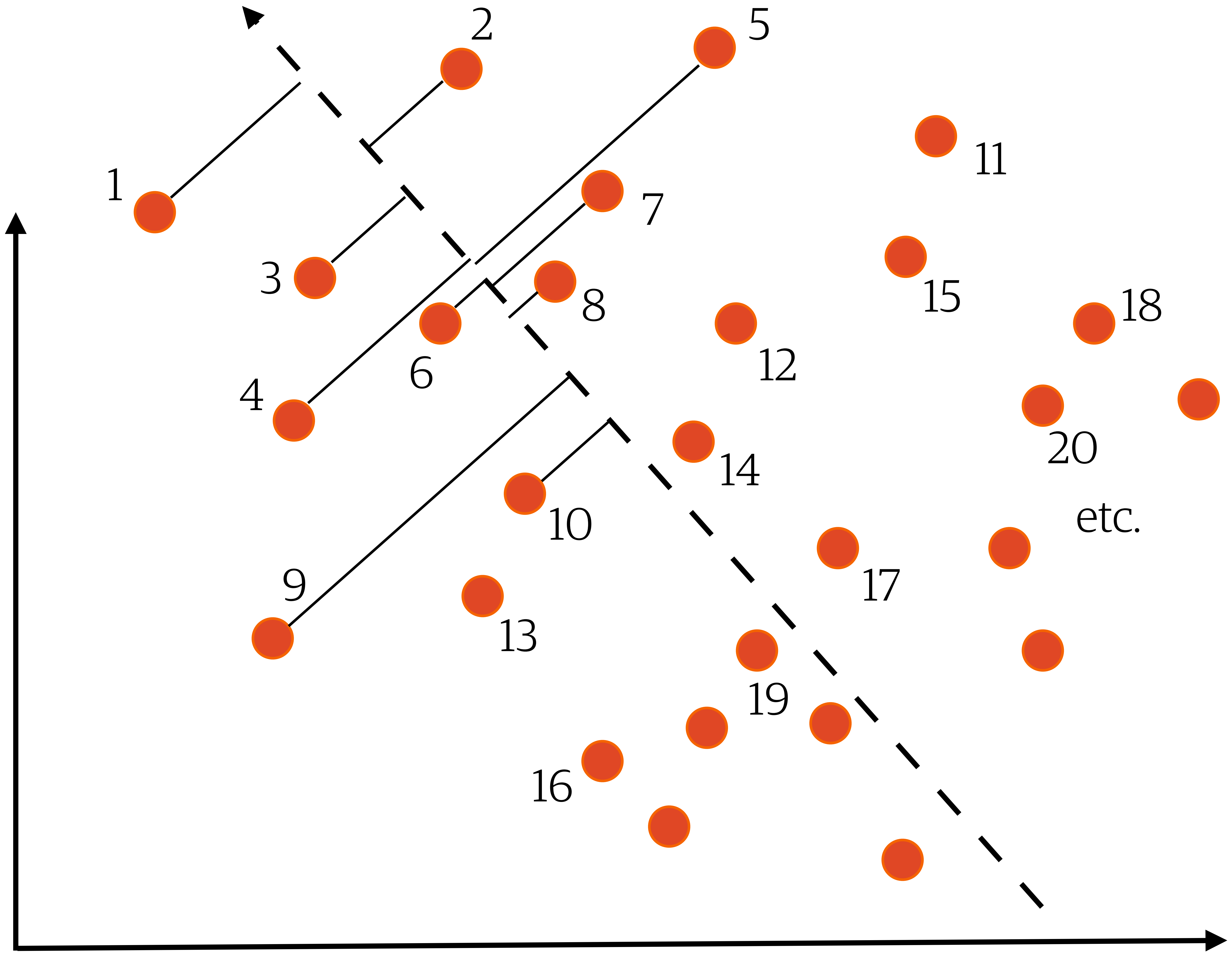
Number of features in model = number of dimensions in “feature space”

Support Vector Machines for classification



Model weights define a line perpendicular to this boundary

Support Vector Machines for ranking?



Method first presented in "Optimizing Search Engines using Clickthrough Data", T. Joachims, KDD 2003

CONVERT YOUR RANKING PROBLEM INTO A CLASSIFICATION PROBLEM WITH THIS ONE WEIRD TRICK!

Converting to a classification problem

Each training instance represents a *pair* of items from *same* set of search results in your logs.

<item1, item2>

Learner must learn to order item1 and item2 correctly, with respect to user preference decisions found in your logs.

Classifiers need a class label

<item1, item2> ↠ +1

… if user preferred item1 (the winner) to item2 (the loser).

<item1, item2> ↠ –1

… if user preferred item2 to item1.

Concentrate on *differences* between item features

<differences_between_item1_and_item2> ↠ +1

… if user preferred item1 to item2.

<differences_between_item1_and_item2> ↠ –1

… if user preferred item2 to item1.

Q. WHAT'S THE DIFFERENCE BETWEEN TWO VECTORS?
A. LITERALLY JUST SUBTRACTION.

Subtract item2's features from item1's

item1	item2	item1_item2_diff
TITLE_purple ➡ 1.00	TITLE_purple ➡ 1.00	TITLE_purple ➡ +0.00
TITLE_sweater ➡ 1.00	TITLE_sweater ➡ 0.00	TITLE_sweater ➡ +1.00
TITLE_yarn ➡ 0.00	TITLE_yarn ➡ 1.00	TITLE_yarn ➡ –1.00
CLICK_RATE ➡ 0.23	CLICK_RATE ➡ 0.16	CLICK_RATE ➡ +0.07
CONV_RATE ➡ 0.02	CONV_RATE ➡ 0.05	CONV_RATE ➡ –0.03
PRICE_QTILE ➡ 0.73	PRICE_QTILE ➡ 0.39	PRICE_QTILE ➡ +0.34

Train on these differences

item1_item2_diff

TITLE_sweater ➡ +1.00
TITLE_yarn ➡ –1.00
CLICK_RATE ➡ +0.07
CONV_RATE ➡ –0.03
PRICE_QTILE ➡ +0.34

label ↠ +1

"Please learn that these feature differences are associated with item1 winning and item2 losing."

Train on these differences

item1_item2_diff

TITLE_sweater ➡ +1.00
TITLE_yarn ➡ –1.00
CLICK_RATE ➡ +0.07
CONV_RATE ➡ –0.03
PRICE_QTILE ➡ +0.34

label ↠ –1

“Please learn that these feature differences are associated with item2 winning and item1 losing.”

Apply model to *individual* items

Intuition:

Item's score is *positively* affected by having features that are often found in the "winner" of a preference decision.

It's *negatively* affected by having features that are often found in the "loser" of a preference decision.

Item

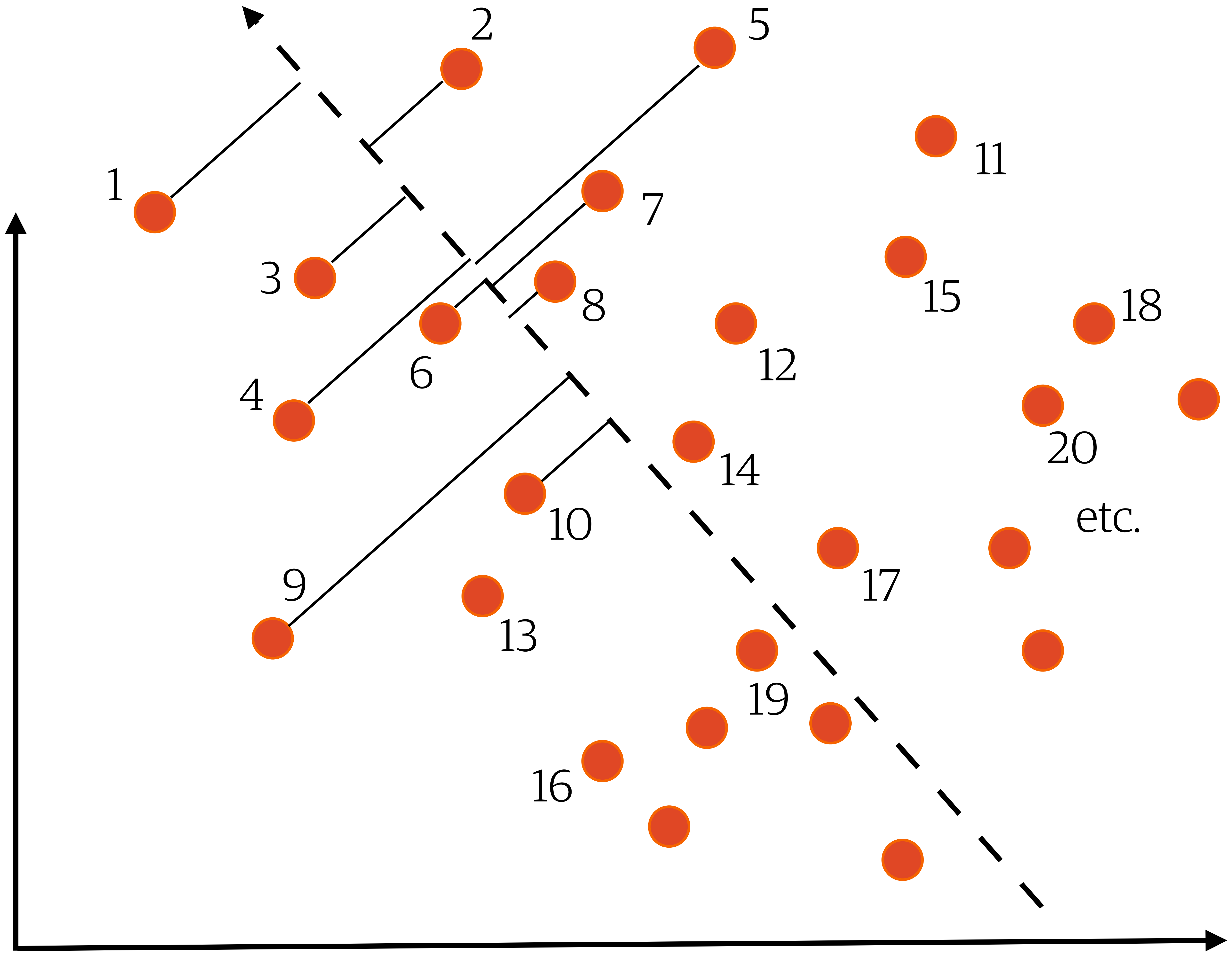
weighted sum

Model

TITLE_hair ➡ 0.25
TITLE_hand ➡ 0.09
TITLE_purple ➡ 0.31
TITLE_sweater ➡ 0.28
USER_CLICKED ➡ 1.00
AGE_YEARS ➡ 0.10

TITLE_hair ➡ +0.01
TITLE_hand ➡ +0.03
TITLE_purple ➡ +0.14
TITLE_sweater ➡ +0.08
USER_CLICKED ➡ +0.46
AGE_YEARS ➡ –0.12
...

Order by this score to reconstruct ranking



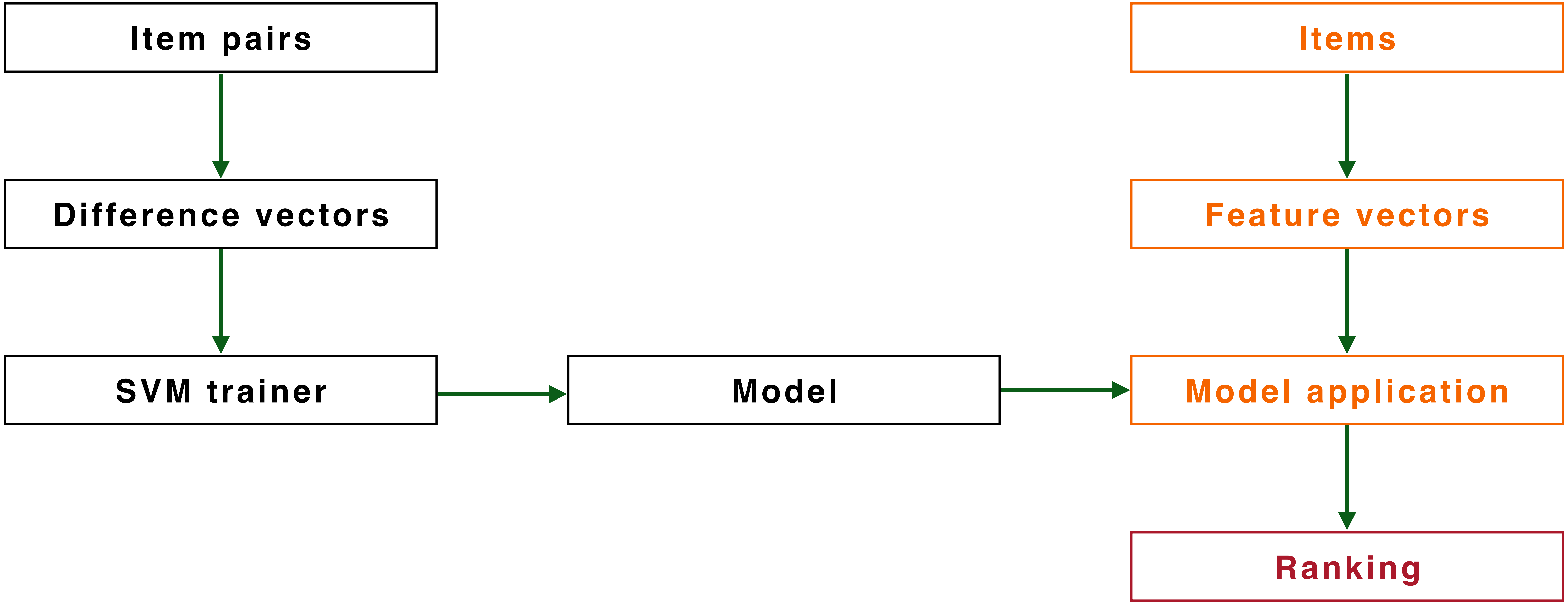
See also: "Large Scale Learning to Rank", D. Sculley, NIPS 2009 Workshop on Advances in Ranking

SECTION 4

LTR in production

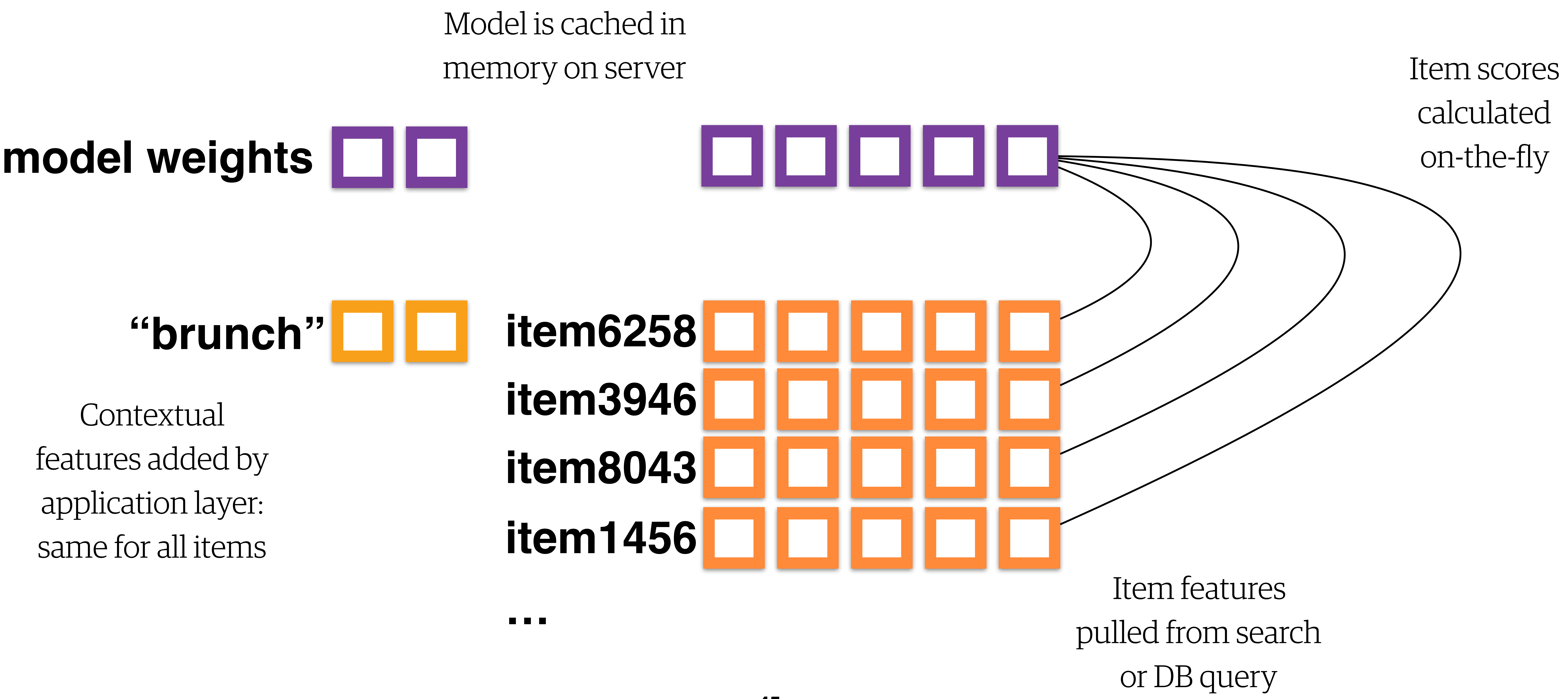
WHEN AND WHERE SHOULD YOU CALCULATE THESE RANKING SCORES?

Precomputing them offline is simplest



Might be unfeasible for all items/all queries

Dynamic ranking when user runs query



Dynamic top-K reranking

Get initial result set from simpler method:

e.g. traditional search query.

Then build feature vectors and calculate scores for top results only. (Top 10, 100, 1000…)

See also "Learning to Rank in Solr", Nilsson & Ceccarelli, Lucene/Solr Revolution 2015

Before	After
ID1375 rel_score=5.7	ID9240 svm_score=1.0
ID8682 rel_score=5.2	ID1375 svm_score=0.9
ID9240 rel_score=5.0	ID8364 svm_score=0.8
ID4173 rel_score=4.6	ID8682 svm_score=0.7
ID8364 rel_score=4.1	ID4173 svm_score=0.6
ID4066 rel_score=3.5	ID4066
ID9246 rel_score=3.4	ID9246
…	…

Top-K reranking:
here K=5.

SOLVING THE RIGHT PROBLEM

Feedback loops and filter bubbles

Make sure you're not just training your model to reinforce existing rankings.

New content needs to get a look-in.

Option: introduce some level of randomization (carefully).

Option: train on one product, apply on another.

Removing position bias

Option: only consider "losers" that were ranked higher than lowest click, when constructing training pairs.

Option: include position as a feature in the model, then set to zero when applying the model.

Option: randomly switch adjacent pairs of search results to remove bias from training data.

Choosing the right target ranking

Make sure the target ranking matches your business need.

Ranking by click alone might be fine for ad placement.

In other contexts, consider taking dwell time or conversion into account. Or, disregard clickbacks and bounces.

A click alone is no guarantee of relevance.

Finer-grained target ranking

Article 78169 ← Articles which were often shared

Article 48016

Article 10945 ← Articles which were often read to end

Article 57297

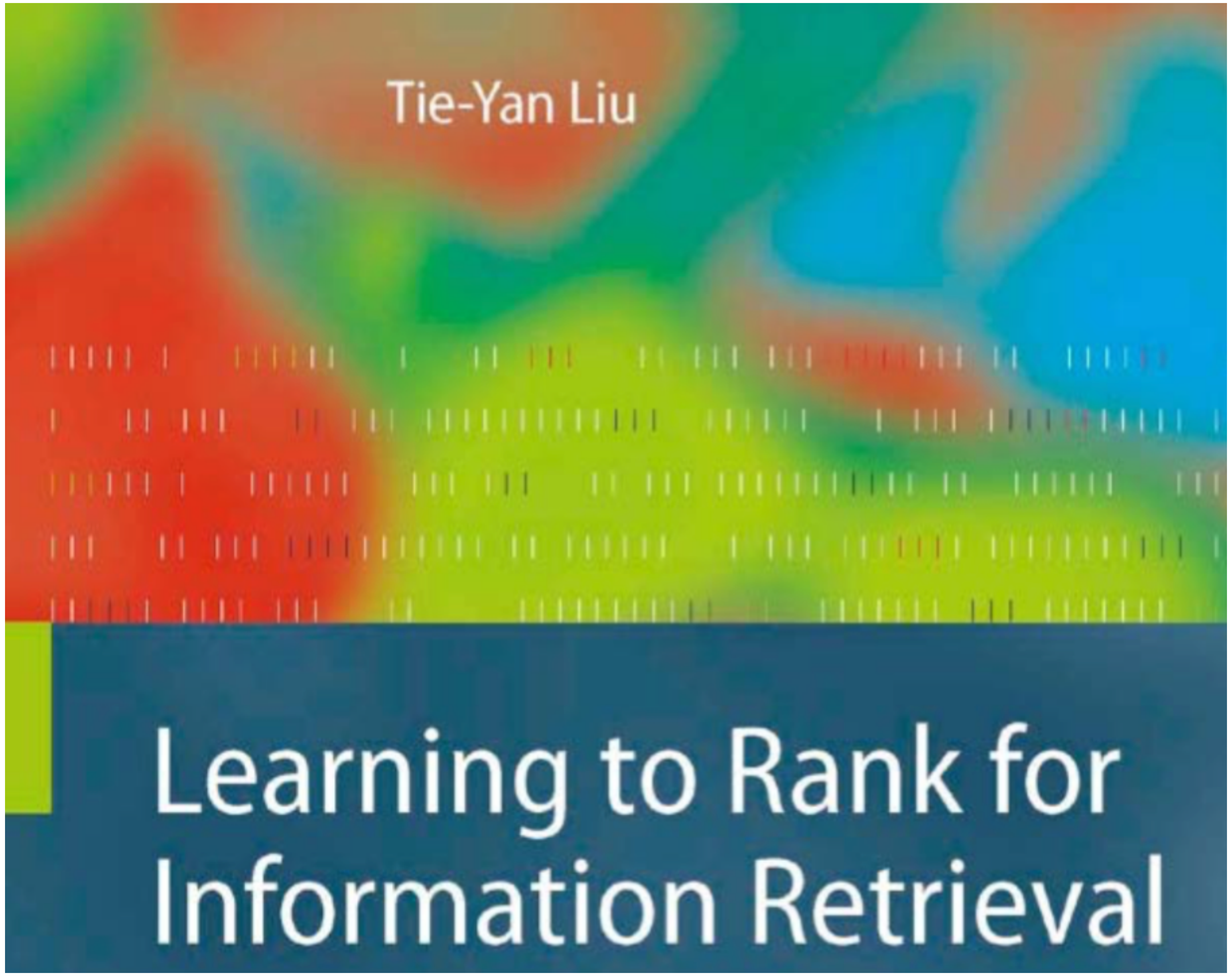
Article 29169

Article 90188 ← Articles which were often clicked

Article 12974

Article 65902

Further reading



Thanks!

TWITTER: @ANDREW_CLEGG

ETSY.COM/CAREERS/