Combinational Collaborative Filtering: An Approach For Personalised, Contextually Relevant Product Recommendation Baskets

Research Project - Jai Chopra (338852)

Dr Wei Wang (Supervisor)

Dr Yifang Sun (Assessor)

Introduction

- Recommendation engines are now heavily used online
- **35%** of Amazon purchases are from algorithms
- We would like to extend on current implementations and provide some more efficient way of generating **goal-oriented** item sets that are **complementary** (i.e. combinational item set recommendations)

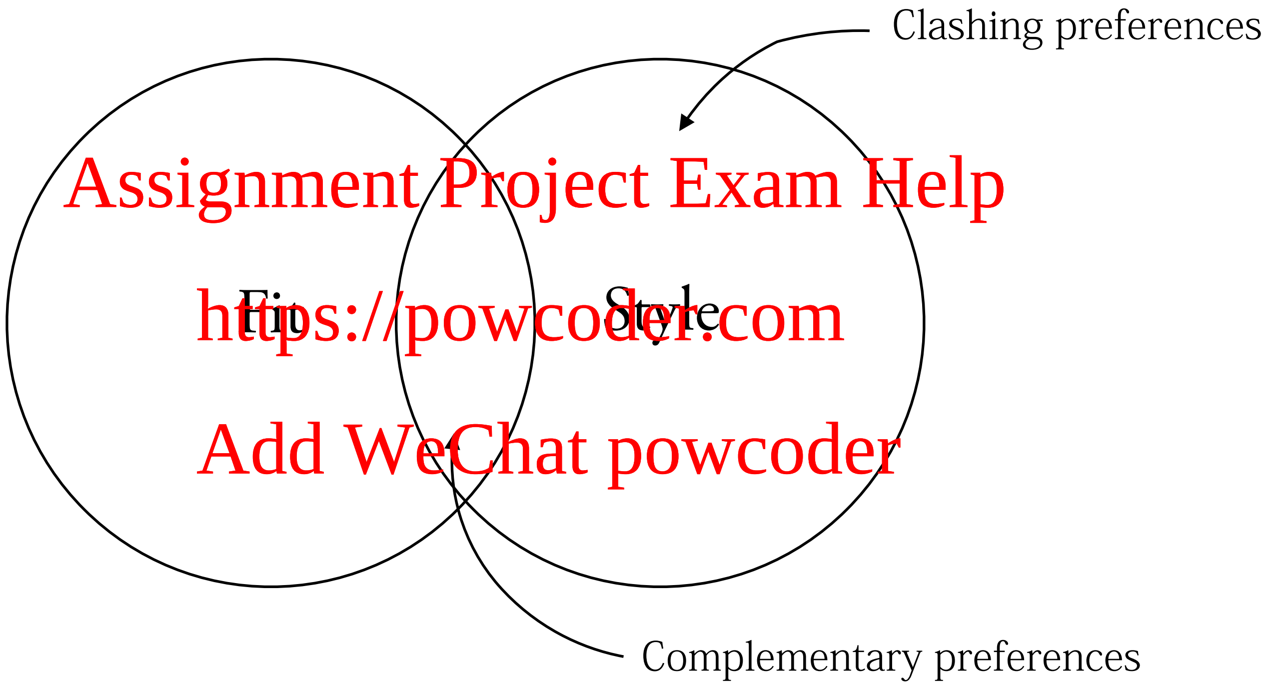
Company background

- Kent and Lime (KAL) is an online styling service
- Data driven business which collects style profile information, feedback and purchase history
- We would like to combine our the KAL dataset, and domain knowledge to produce an autonomous styling agent

Problem space definition

- Goal oriented recommendations
- Contextual recommendations
- Domain knowledge is highly relevant for the design of our system

Problem space definition



Implementation goals

- Recommendations delivered in a timely manner
- Complementary by nature
- Well suited to the customers profile
- Learn and perform performance over time
- Reasonable performance at the start (i.e. avoid cold start)
- Deployed online in a web application environment

Overview of presentation

- Data Preprocessing
- Recommendation engine implementation
- Web application deployment and architecture discussion
- Demonstration
- Experiments and evaluation
- Future considerations

Data preprocessing - version mismatches

- Over time, profile schemas changed
- Solution: pick a subset of data that was common across all schemas

Data preprocessing - missing values

- Many missing values
- Solution: use a mean average, or an initialised value, or discard row

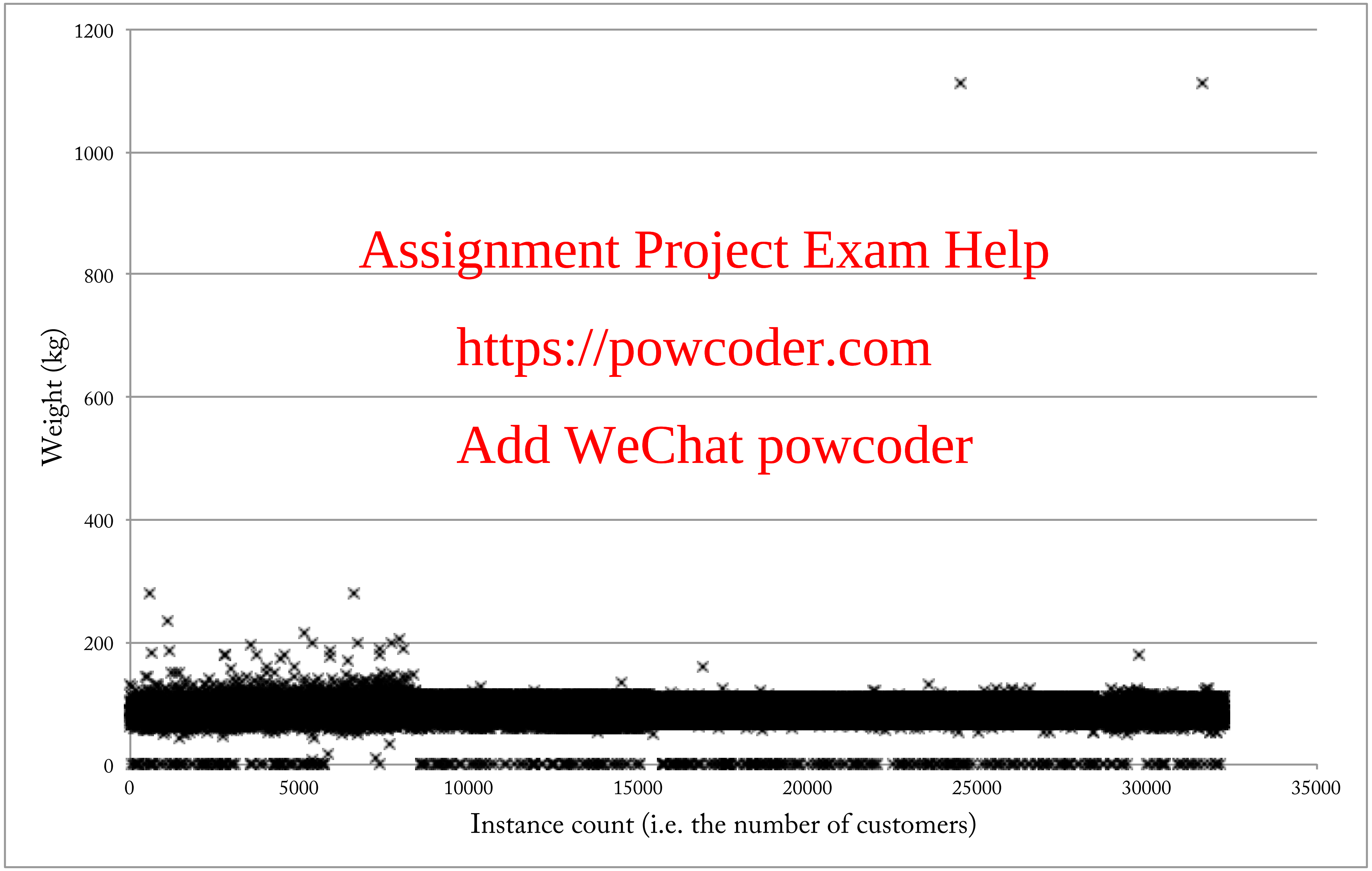
Data preprocessing - inconsistent fields

- Inconsistent values when merging different versions of the schema
- Solution: pick a consistent way, then transform

Data preprocessing - outliers

- Outliers were discovered after the implementation had started, producing highly skewed results
- Retrospectively had to be cleaned (removed) after some analysis

Data preprocessing - outliers



Data preprocessing - initialised state objects

- Define an initialised state for each customer row

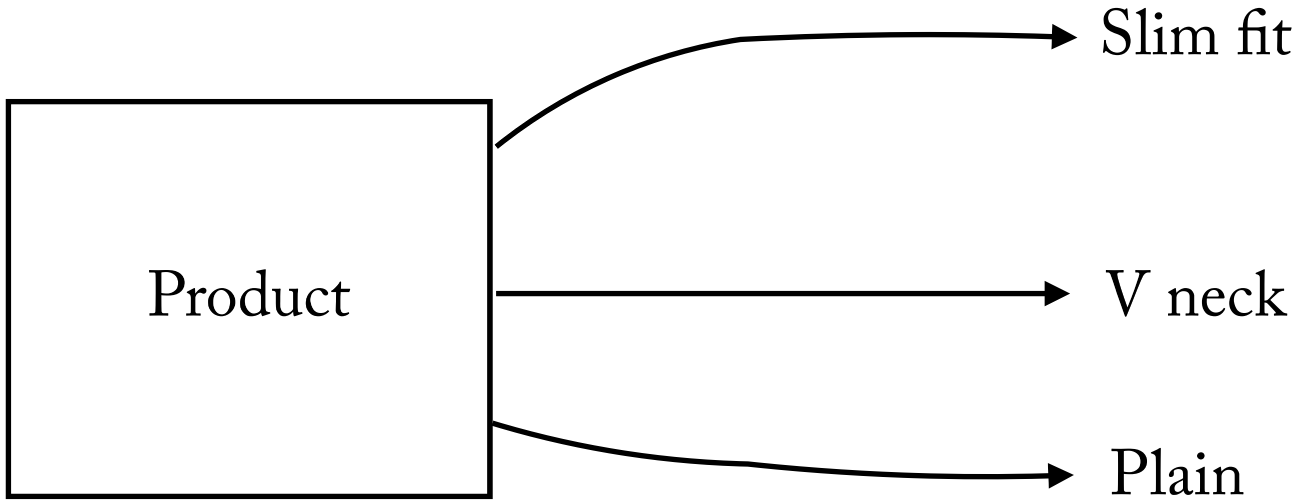
Initialise state: One-hot encoding:

[0,0,0,0] [0,0,1,0]

- Run a check to see, *how dissimilar the row is to the initialised state*. If dissimilarity < threshold, clean/remove row

Data preprocessing - product content data

- Tags are metadata which allow for item to item filtering and some pre-selection
- However, we are unable to perform these on *older* products that do not have tags, thus we filter them out



Implementation (Recommendation Engine)

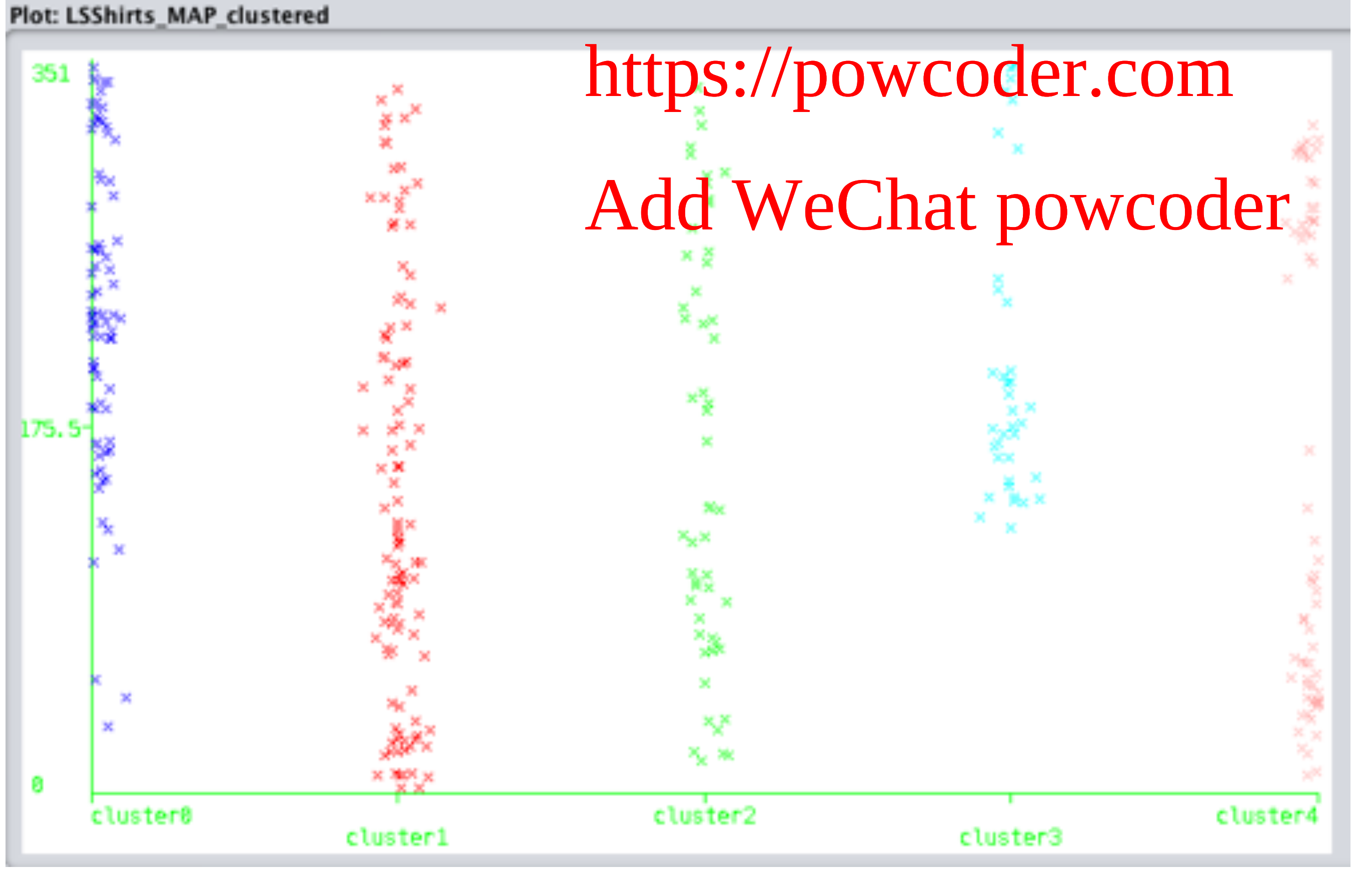
- Xue et al describes two broad types of CF:
  - Memory based approaches
  - Model based approaches
- There are also hybrid approaches, our system would like to use both techniques, primarily memory
- Pennock et al describes a hybrid approach using *'personality type'*, where customers have some pre-selection based on what personality they are

Content boosting using product tags

- Problem: CF will only suggest products that have been purchased previously (bias toward *older* products)
- KAL dataset contains tags, we can use this to perform some clustering based on classification
- Classification solves:
  - Newer products not being selected
  - Evaluation techniques, as it is too difficult to classify on a granular product based level
  - Products are usually out of stock (OOS)

Product clustering design decisions

- What value of K?
  - 5 was selected after analysing results with inventory staff



Clustered Instances

0	78	(22%)
1	113	(32%)
2	53	(15%)
3	37	(11%)
4	71	(20%)

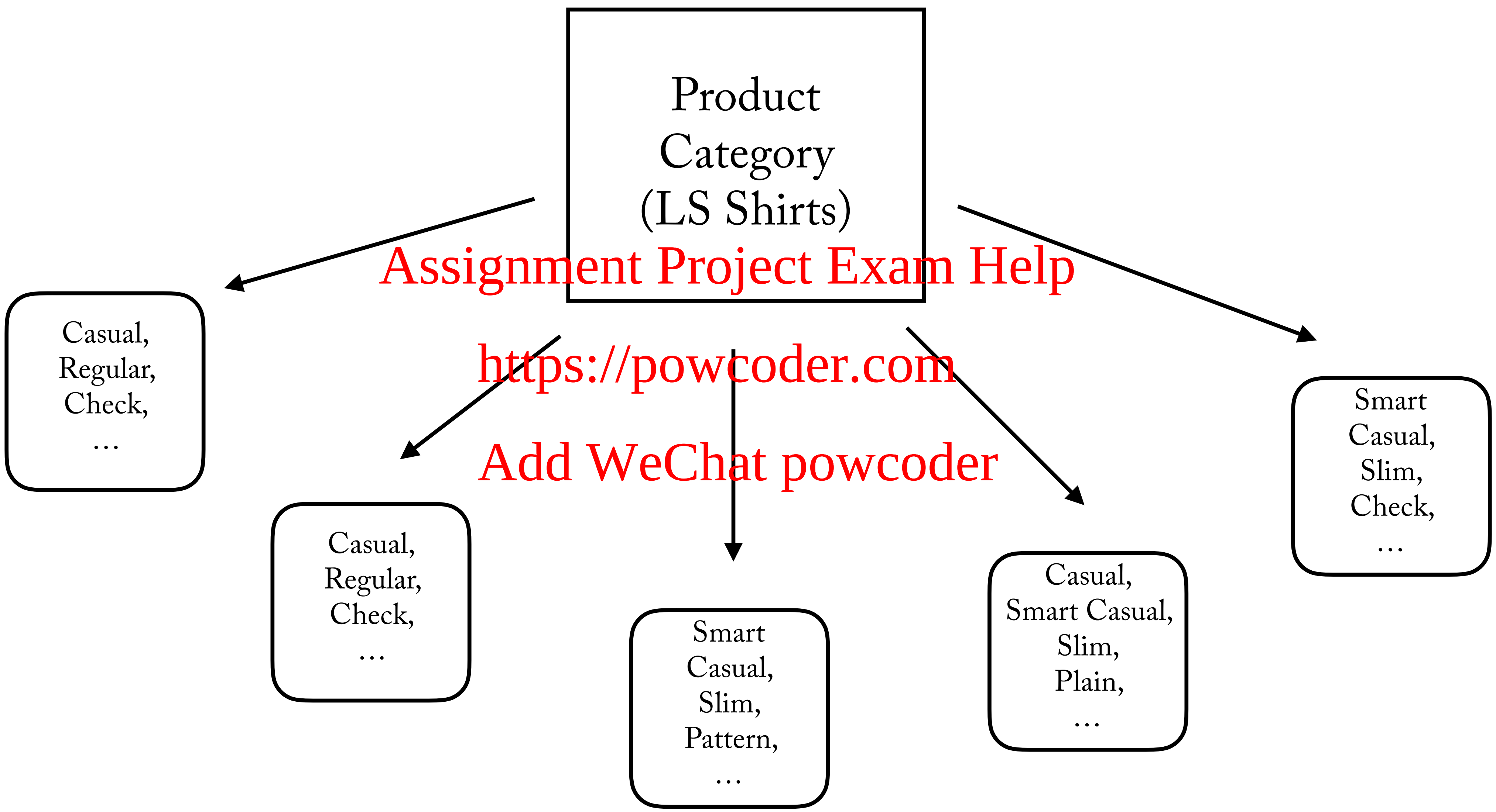
Product clustering design decisions

Final cluster centroids:

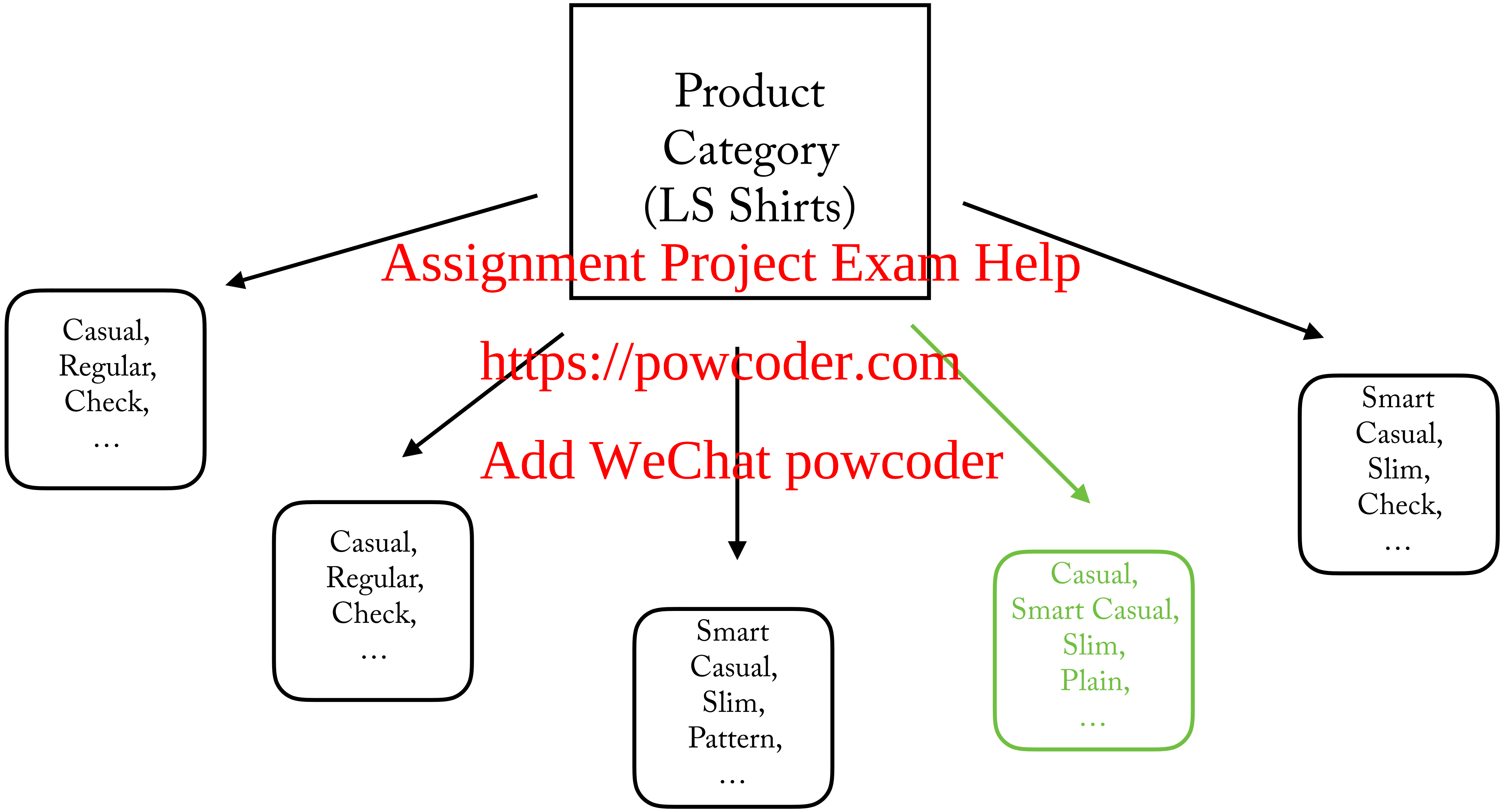
Attribute	Full Data	Cluster# 0	1	2	3	4
	(352.0)	(78.0)	(113.0)	(53.0)	(37.0)	(71.0)
Casual10	1	1	0	1	0	1
SmartCasual11	1	0	1	1	1	0
Dress12	0	0	0	0	0	0
Regular13	1	1	0	0	0	1
Slim14	0	0	1	1	1	0
Check15	0	1	0	0	1	1
Pattern16	0	0	1	0	0	0
Stripe17	0	0	0	0	0	0
Plain18	0	0	0	1	0	0

Cluster 0: {Casual, Regular, Check}

Goal is to select product class, not product



Goal is to select product class, not product



How do we determine winning class?

- Problem: We wish to find the winning class
- Naively, we can find the winning product and abstract this to its class, but this is flawed
- Thus, we wish to find the average score for each class, and pick the highest one

Data: List of product categories matrices P, with length l
Result: Highest scoring product category (or class)
$p_{max} \leftarrow 0$;
p_{best};
for $i \leftarrow 0$ **to** l **do**
  $p_{score} \leftarrow$ averageProductScore($P[i]$);
  if $p_{max} < p_{score}$ **then**
    $p_{max} \leftarrow p_{score}$;
    $p_{best} \leftarrow P[i]$;
  end
end

Algorithm 1: Determine the winning product classification from the average product score in each cluster

Problems with selecting winning class

- Dominant product scores
  - Products that are older have more rating (time is biased)
  - Products that new new have no rating
- Sparsity
  - No rating products can thus influence the class selection

Thresholding dominant product scores

- Use a generalised logistic function (modified)

$$f(x) = \frac{20}{(1 + 2e^{-2x})^{1/2}}$$

Smoothing for “unrated” products

- Common technique used to reduce sparsity
- Select the average score of the cluster and associate with unrated products

Noise reduction

- Remove customers that have not purchased more than 2 times (i.e. 2 baskets)

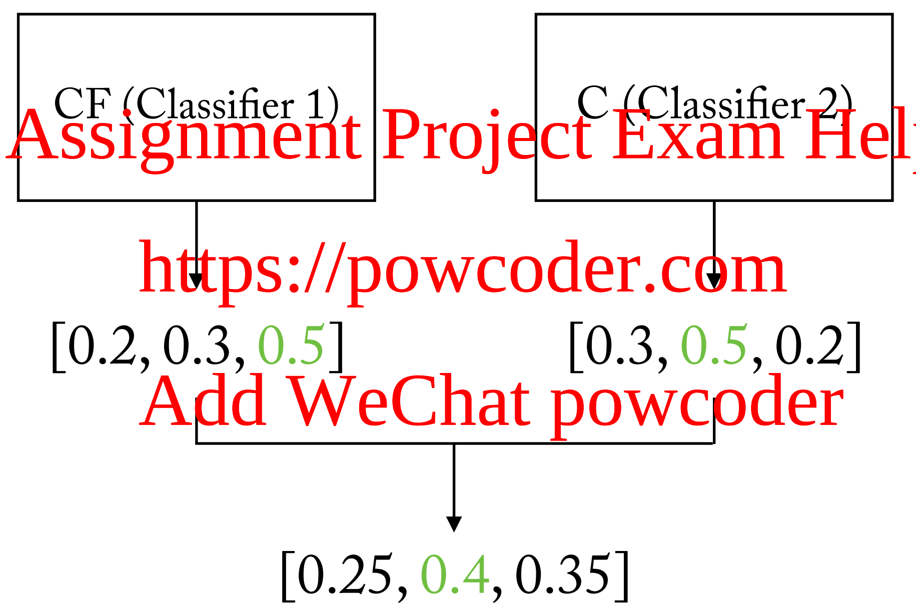
“Slow start” problem of CF

- Customers that have never purchased before have poor system accuracy

- How do we resolve this? We can build a model where similarity is built on customer profiling, rather than purchase history

- We can then combine “votes” for product classifications in an ensemble classifier

“Slow start” problem of CF



"Slow start" problem of CF

- Introduce a weighted "voter" function, and treat each category classification as a probability. Similar to MLE.

$$\arg\max_x f(x) = \lambda.AvgScore_{C_{i_{cf}}} + (1 - \lambda).AvgScore_{C_{i_c}}$$

Self weighting

- We also want to vote the *active* users votes higher than its peers in the neighbourhood, if it has rated products or given feedback

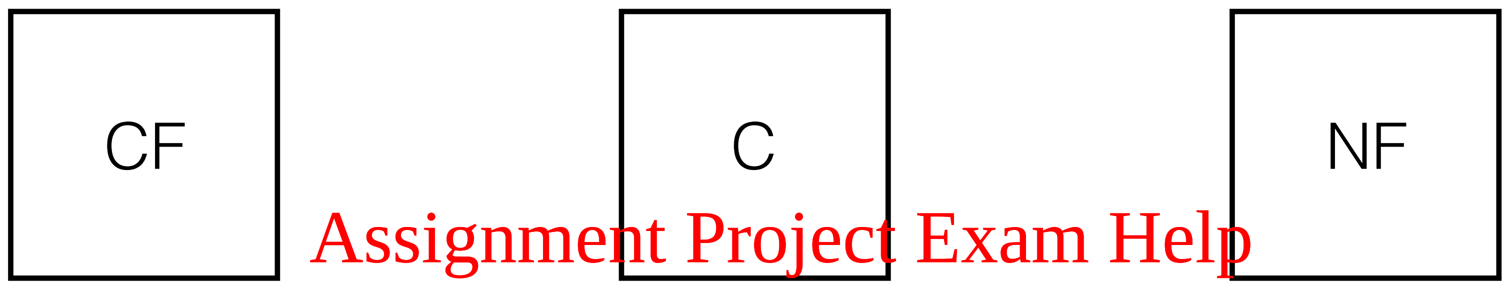
Design decisions - producing customer clusters

- Initially k-means was unable to produce some nice clusters that reflected both our fit and style knowledge domains
- We prefer to weight physical attributes higher over non-physical attributes (as this is what a stylists *naturally* does)
- We also *increase k* to capture both style knowledge domains

Feedback data incorporation

- KAL dataset contains negative explicit feedback based on style, as well as negative implicit feedback based on fit
- Author chose to discard fit based feedback (too noisy/ uncertain)
- Explicit feedback was used to *rate product classifications*, and then subtract a value from the final vote, i.e. add to the ensemble method

Ensemble voting classifier



Output votes for product class
Select the highest product class

Building combinational (complementary) baskets

- Use rule mining to build IF-THEN statements so we can modify our baskets before our final item sets
- Look for common trends between product classifications e.g. If `LSShirts_1, Shorts_2 => Socks_3`

Web architecture design decisions

- The goal of the project is to implement an efficient recommendation system, thus the web application itself should be *fast*

Node streams and HTTP chunks

- Use data frames to split a product list into products
- Operate on each product, rather than a data frame
- Send the product to the client while the next data frame is processing

Optimising database queries

- Filter noise (i.e. only query customers with purchases > 2)
- Filter OOS products (useful for item to item filtering)
- MongoDB Pipeline aggregation queries

Online / incremental updates

- Once a customer has given feedback, we wish to suggest them a new item immediately
- Thus, we want our model to be *online*, that is, it updates incrementally
- With instance based learning techniques this is possible

Experiments and evaluation

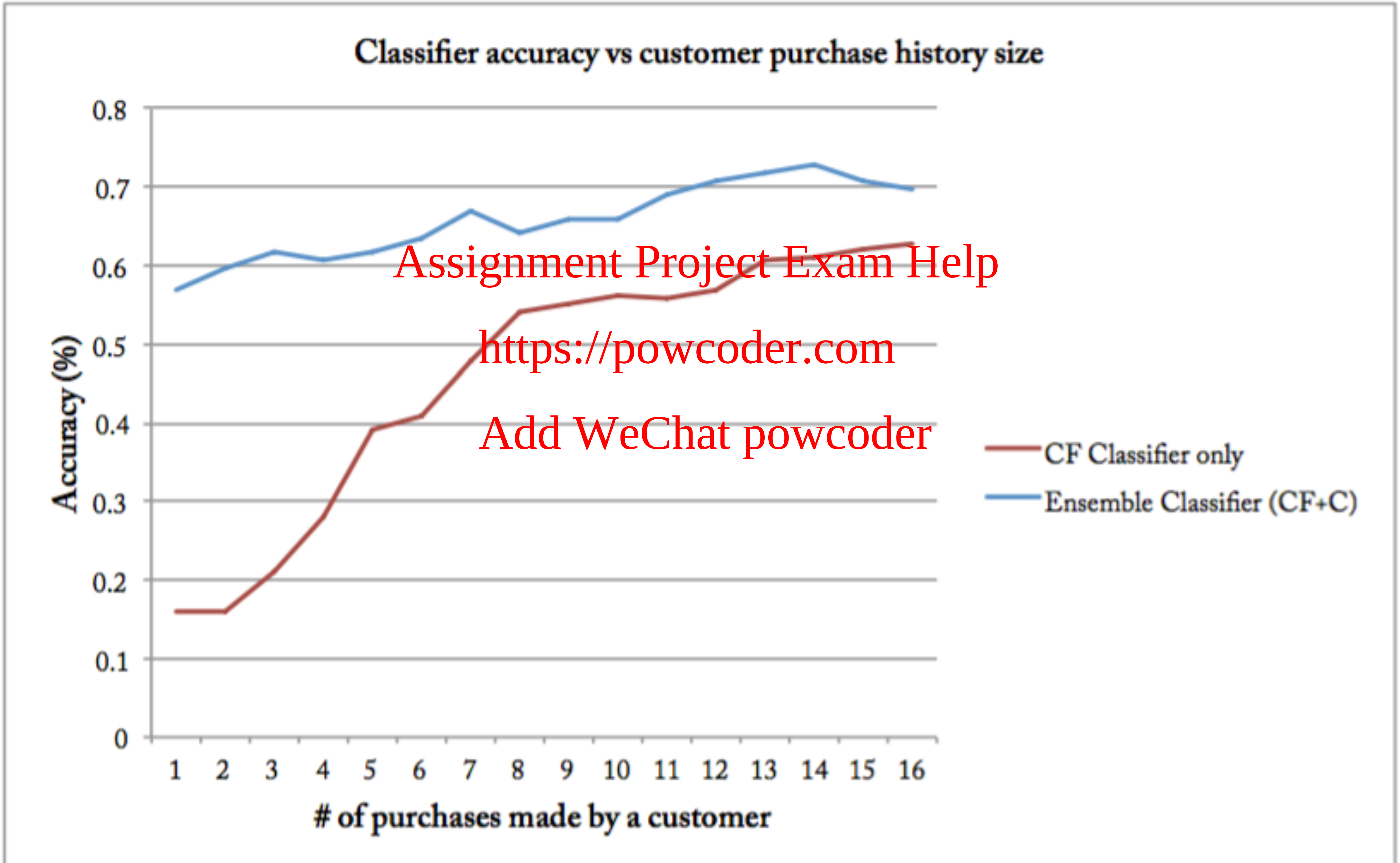
- Experiments:
  - Does our ensemble method increase accuracy? What value of lamda do we use?
  - Optimisation methods: smoothing, self weighting, reducing noise impact
  - Speed of transfer in web

Experiments and evaluation

- Measure to use, mean absolute error (MAE)

$$\frac{1}{n}\sum_{i}\epsilon_i$$

Experiments and evaluation



Experiments and evaluation

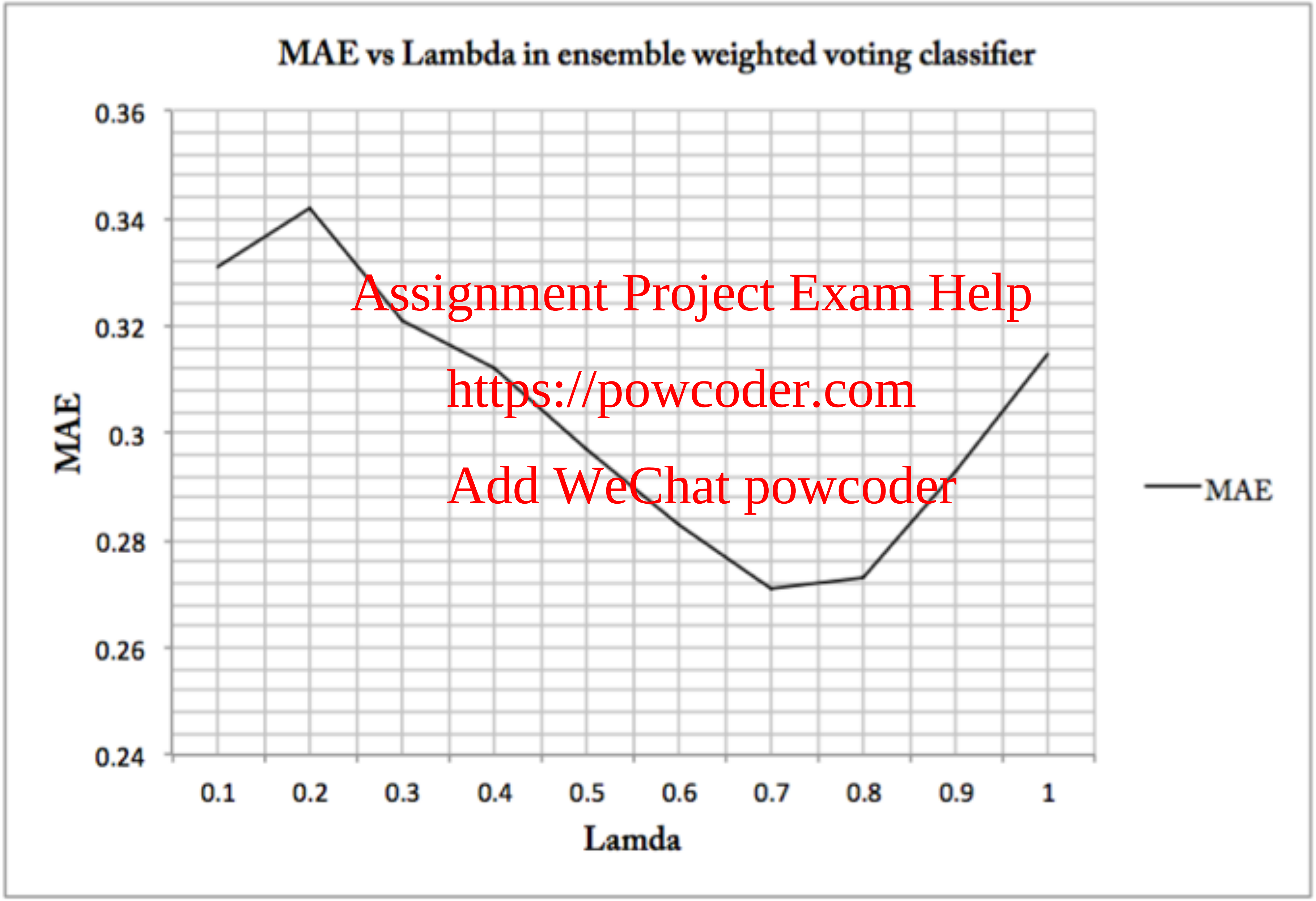


Figure 7.1: Minimising MAE with respect to lamda in the ensemble classifier, CF + C + NF

Experiments and evaluation

*	Ensemble method			
Optimisations	PCF	PCF + C	PCF + C + NF	PCF + C + NF + RS
SW	0.411	0.353	0.356	0.313
SW + FN	0.394	0.298	0.276	0.274
SW + FN + SG	0.392	0.292	0.274	**0.271**

Table 7.1: Mean Absolute Error (MAE) performance for various ensemble method recommendation systems

Production results

- Deployed to production on 18 November

Metric captured	Value
Negative feedback per customer	2.36 (avg)
Positive feedback per customer	8.60 (avg)
Average latency response time	2302 (ms)
Percentage of customers that complete	84%
Total ad-hoc feedback collected	3110

Table 7.3: Metrics captured from the recommendation system in a live production enviroment, from Nov 18 to 24th

Future work and considerations

- Implement new rules based on stylist domain knowledge with respect to product attributes: colour, seasonality, time-variant features

- Filter categorical products for non-relevant seasons
- Testing memory-based CF vs model-based CF (i.e. given training set, build model, probabilistic measure of likely product classification)
- Explore other research based methods, genetic algorithms

Conclusion

- Very interesting project
- Many different fields of CS - Machine learning, data mining, data warehousing, web application development, human centered design
- Ground work for future potential in this area
- Thank you for listening