Machine Learning

Building recommendation systems in Python with LightFM

Petrus Janse van Rensburg

Machine Learning

Using LightFM to run a dairy farm in Tanzania

Petrus Janse van Rensburg

Machine Learning

Using LightFM to ~~run a dairy farm in Tanzania~~ recommend groceries in Cape Town

Petrus Janse van Rensburg

NETFLIX

Spotify®

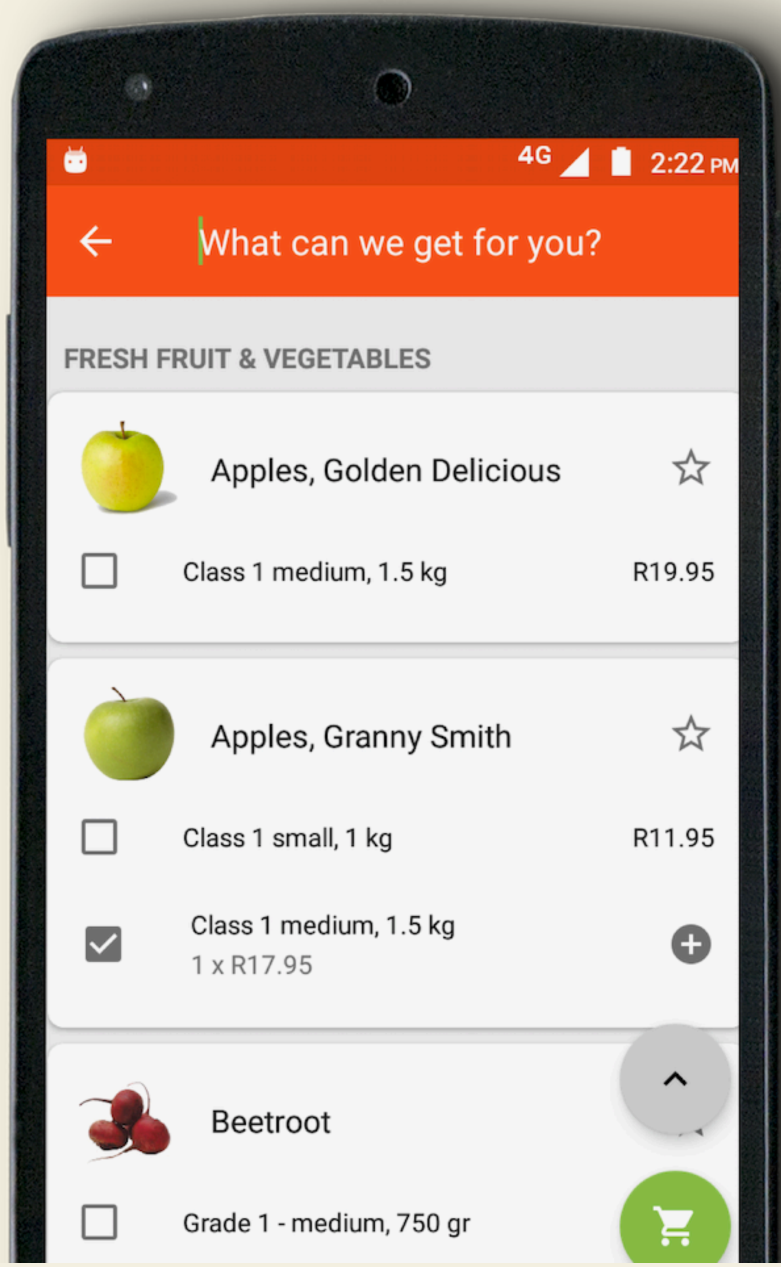
UBER
eats

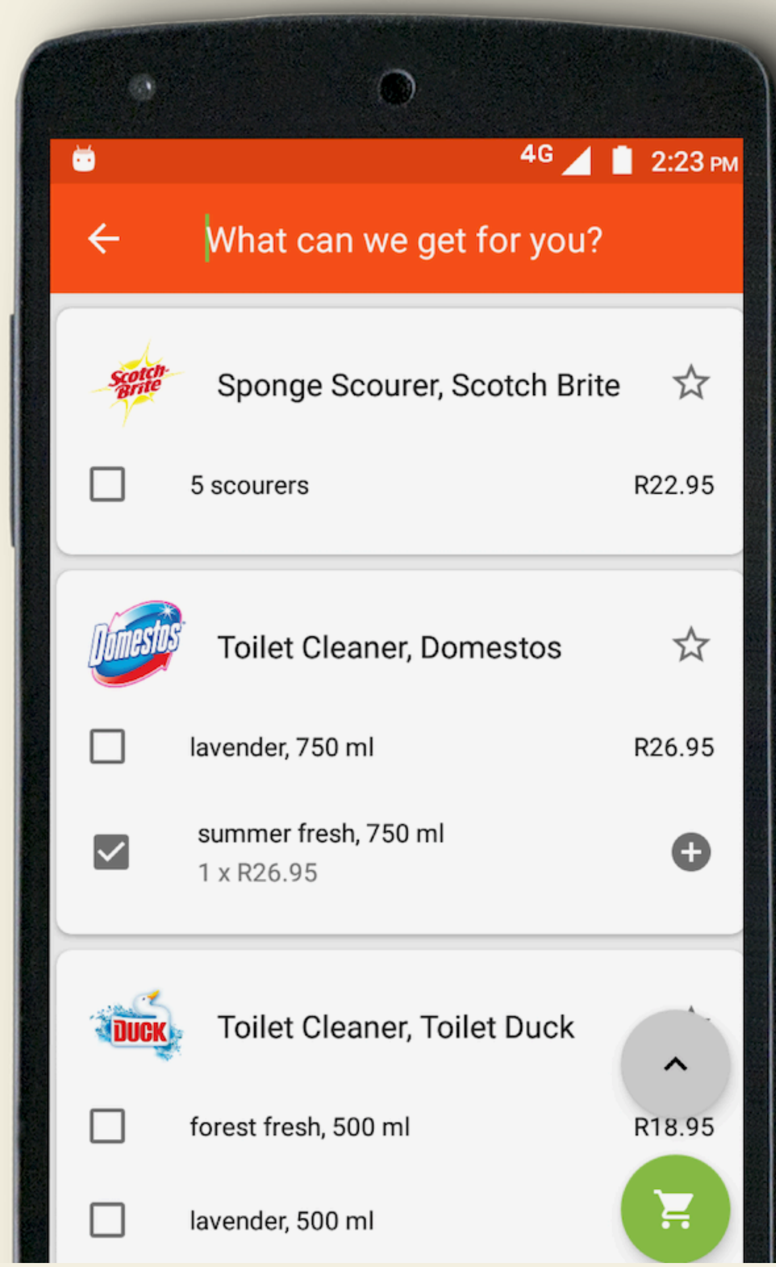
amazon

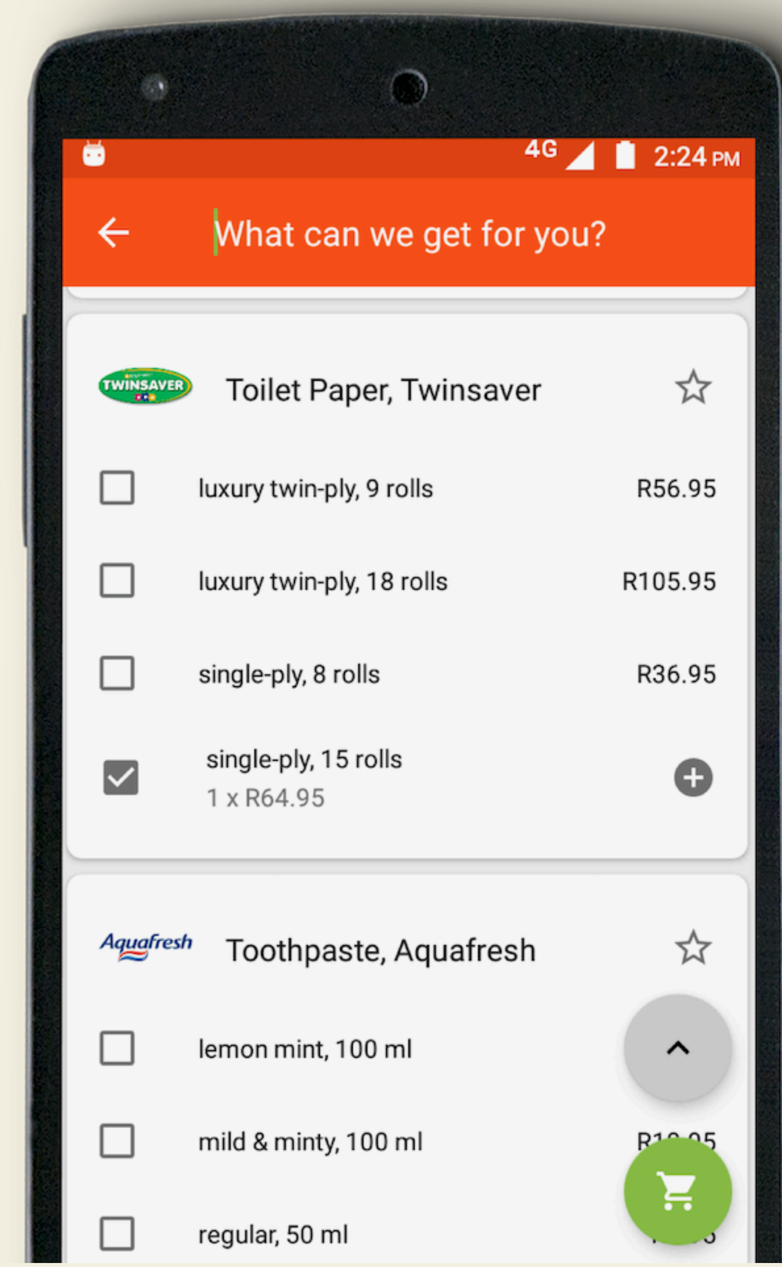
“People who liked this item, also liked…”

LightFM

- Incorporates user-item interaction data + feature information into the same model
- E.g. collaborative + content-based filtering
- Uses a traditional matrix-factorisation approach (rather than a neural-network approach)
- Useful for a wide range of problems (movies, restaurants, groceries)


4G 2:22 PM
What can we get for you?
FRESH FRUIT & VEGETABLES
Apples, Golden Delicious
Class 1 medium, 1.5 kg
R19.95
Apples, Granny Smith
Class 1 small, 1 kg
R11.95
Class 1 medium, 1.5 kg
1 x R17.95
Beetroot
Grade 1 - medium, 750 gr


4G 2:23 PM
What can we get for you?
Sponge Scourer, Scotch Brite
5 scourers
R22.95
Toilet Cleaner, Domestos
lavender, 750 ml
R26.95
summer fresh, 750 ml
1 x R26.95
Toilet Cleaner, Toilet Duck
forest fresh, 500 ml
R18.95
lavender, 500 ml


4G 2:24 PM
What can we get for you?
Toilet Paper, Twinsaver
luxury twin-ply, 9 rolls
R56.95
luxury twin-ply, 18 rolls
R105.95
single-ply, 8 rolls
R36.95
single-ply, 15 rolls
1 x R64.95
Toothpaste, Aquafresh
lemon mint, 100 ml
mild & minty, 100 ml
regular, 50 ml

“It’s tough to make predictions, especially about the future”

– *Neils Bohr and others*

Collaborative filtering

User-item interaction matrix

$$I = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & \dots & 1 & 0 & 1 \\ 0 & 1 & 0 & \dots & 1 & 0 & 0 \end{bmatrix}$$

Calculate cosine distances to find similar users and make recommendations based on their interactions

Collaborative filtering

$$
I = \begin{array}{cl}
\begin{array}{ccccccc} \text{five roses tea} & \text{huggies diapers} & \text{iwisa maize meal} & & \text{knorrox stock} & \text{oros squash} & \text{tomatoes} \end{array} & \\
\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 1 & 0 & 1 \\ 0 & 1 & 0 & \ldots & 1 & 0 & 0 \end{bmatrix} & \begin{array}{l} \text{user 1} \\ \text{user 2} \\ \text{user 3} \end{array}
\end{array}
$$

Calculate cosine distances to find similar users and make recommendations based on their interactions

Matrix factorization

Low-dimensional user- and item embeddings

$$\underset{(users \times items)}{I} \approx \underset{(users \times n)}{U} \; \underset{(n \times items)}{P}$$

simple multiplication to predict scores

Matrix factorization

Dot-product between user and item representations

$$score\ 1 = \underset{(user\ 1)}{\begin{bmatrix} 0.25 & 0.93 & 1.43 & 0.48 & 0.34 \end{bmatrix}} \underset{(item\ 1)}{\begin{bmatrix} 0.53 \\ 0.34 \\ 0.53 \\ 0.82 \\ 1.37 \end{bmatrix}}$$

Matrix factorization

- You can decide how many dimensions to use for your embeddings.
- Factorization can be done in a number of ways, e.g.
    - Stochastic Gradient Descent
    - Alternating Least Squares

Shortcomings

- Some users haven’t bought anything.
- Some products haven’t been bought.

Content-based filtering

Make use of item feature information

$$['huggies', 'diapers']$$

$$['huggies', 'nappy\ pants']$$

$$['huggies', 'baby\ wipes']$$

$$['pampers', 'diapers']$$

One-hot encode item features, then calculate cosine distances to find similar items and make recommendations

One-hot encoding

Vectorize categorical variables

$$\begin{array}{r c c c c c c}
 & & \text{huggies} & \text{pampers} & \text{diapers} & \text{nappy pants} & \text{baby wipes} \\
item\ 1 & = & [1 & 0 & 1 & 0 & 0] \\
item\ 2 & = & [1 & 0 & 0 & 1 & 0] \\
item\ 3 & = & [1 & 0 & 0 & 0 & 1] \\
item\ 4 & = & [0 & 1 & 1 & 0 & 0]
\end{array}$$

Shortcomings

- Doesn’t work for new users.
- No information-sharing between users.

Hybrid approach

user-item interactions

+

item features

+

user features

Hybrid approach

Users can also have features

$$\left['Cape\ Town', 'Observatory'\right]$$

$$\left['Cape\ Town', 'Mitchell's\ plain'\right]$$

$$\left['Cape\ Town', 'Langa'\right]$$

$$\left['Cape\ Town', 'City\ Center'\right]$$

One-hot encoding

Vectorize categorical variables

$$
\begin{array}{rccccc}
 & \text{cape town} & \text{observatory} & \text{mitchell's plain} & \text{langa} & \text{city center} \\
user\ 1 = & [1 & 1 & 0 & 0 & 0] \\
user\ 2 = & [1 & 0 & 1 & 0 & 0] \\
user\ 3 = & [1 & 0 & 0 & 1 & 0] \\
user\ 4 = & [1 & 0 & 0 & 0 & 1]
\end{array}
$$

LightFM

How it works?

- Matrix Factorization that can incorporate interactions, item features and user features.
- Feature representations are summed before taking a dot-product for calculating score.
- Reduces to regular MF model when no features are provided.

LightFM

Matrix Factorization with Stochastic Gradient Descent

- Loss Function

    Bayesian Personalised Ranking (BPR)

    Weighted Approximate-Rank Pairwise (WARP)

- Learning Rate
- No. of components in latent space

LightFM

$$\underset{(user\ features \times item\ features)}{I} \approx \underset{(user\ features \times n)}{U} \underset{(n \times item\ features)}{P}$$

addition, then multiplication to predict scores

LightFM

Addition, then multiplication to predict scores

$$
\begin{aligned}
user\ 1 &= western\ cape + cape\ town + observatory \\
item\ 1 &= huggies + diapers
\end{aligned}
$$

$$score = user\ 1 \cdot item\ 1$$

Testing / evaluation

Split your interactions dataset, keep one part out for testing.
The goal is to not have a large discrepancy between train and test sets (otherwise you're overfitting).

1. AUC ROC score:
   (Area-Under-Curve Receiver Operating Characteristic)
   The probability that a randomly chosen positive example has a higher score than a randomly chosen negative example. A perfect score is 1.0

2. Precision at k score:
   The fraction of known positives in the first k positions of the ranked list of results. A perfect score is 1.0.

Grocery recommendation example

Assumptions

1. The future will be like the past.

2. Synonymy is accounted for when extracting features e.g.

    - “bath soap” vs. “beauty soap”

    - “peanut butter” vs. “butter”

3. Time-sensitivity is accounted for.

Further reading

- LightFM ‘examples’ directory on GitHub
- Maciej Kula - PyData 2016 talk
- Maciej Kula article on arxiv: “Metadata Embeddings for User and Item Cold-start Recommendations”
- Spotlight package, if you’d rather use neural network / deep-learning approach

Thank you

petrus.jvrensburg@gmail.com
@petrusjvr