join at Slido.com with #bigdata2018

BIG DATA CONFERENCE.

datatonic

Deep Learning for Recommender Systems

Oliver Gindele
@tinyoli
oliver.gindele@datatonic.com

Big Data Conference Vilnius
28.11.2018

Who is Oliver?

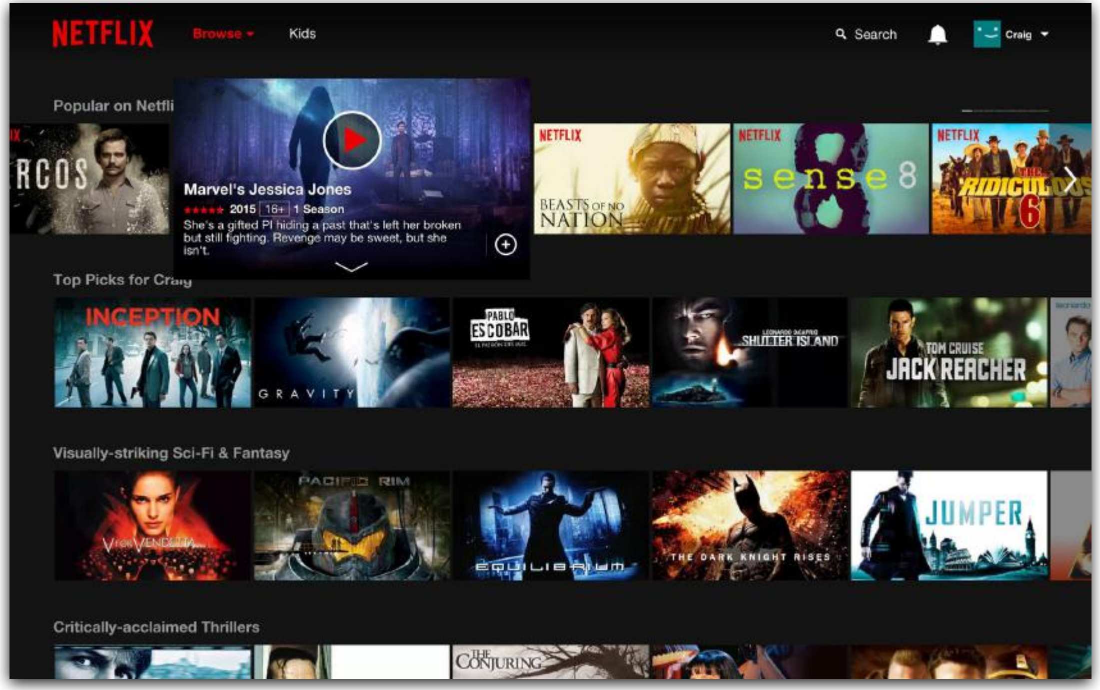
+ Head of Machine Learning
+ PhD in computational physics

Who is datatonic?

We are a **strong team** of **data scientists, machine learning experts, software engineers** and **mathematicians.**

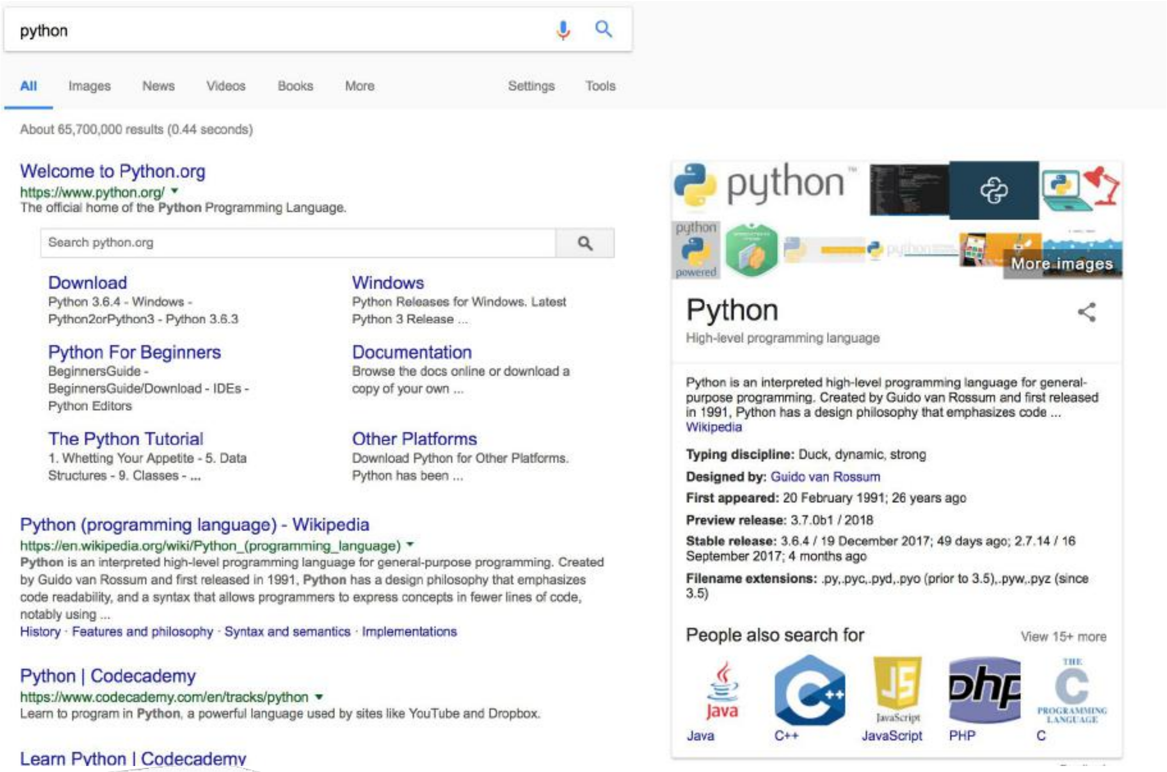
Our mission is to **provide tailor-made** systems to help your organization get **smart actionable insights** from **large data volumes.**

Recommender Systems

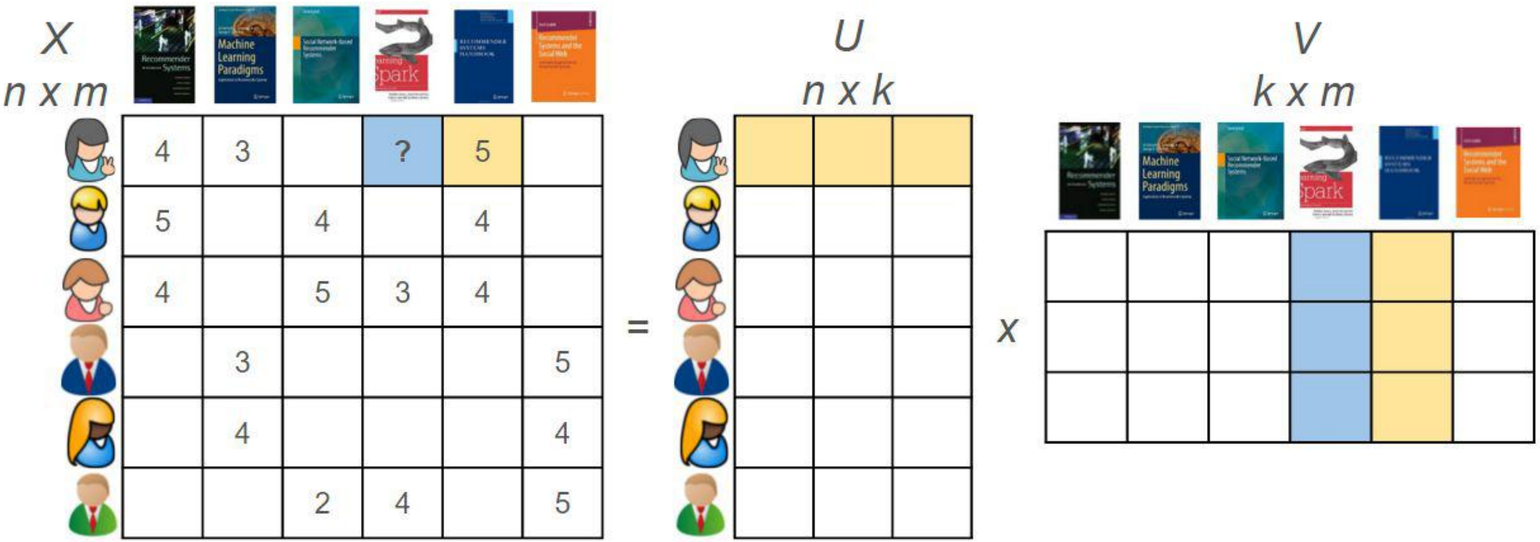


Recommender Systems

Recommender Systems



Collaborative Filtering - Introduction



Collaborative Filtering - Introduction

Objective:

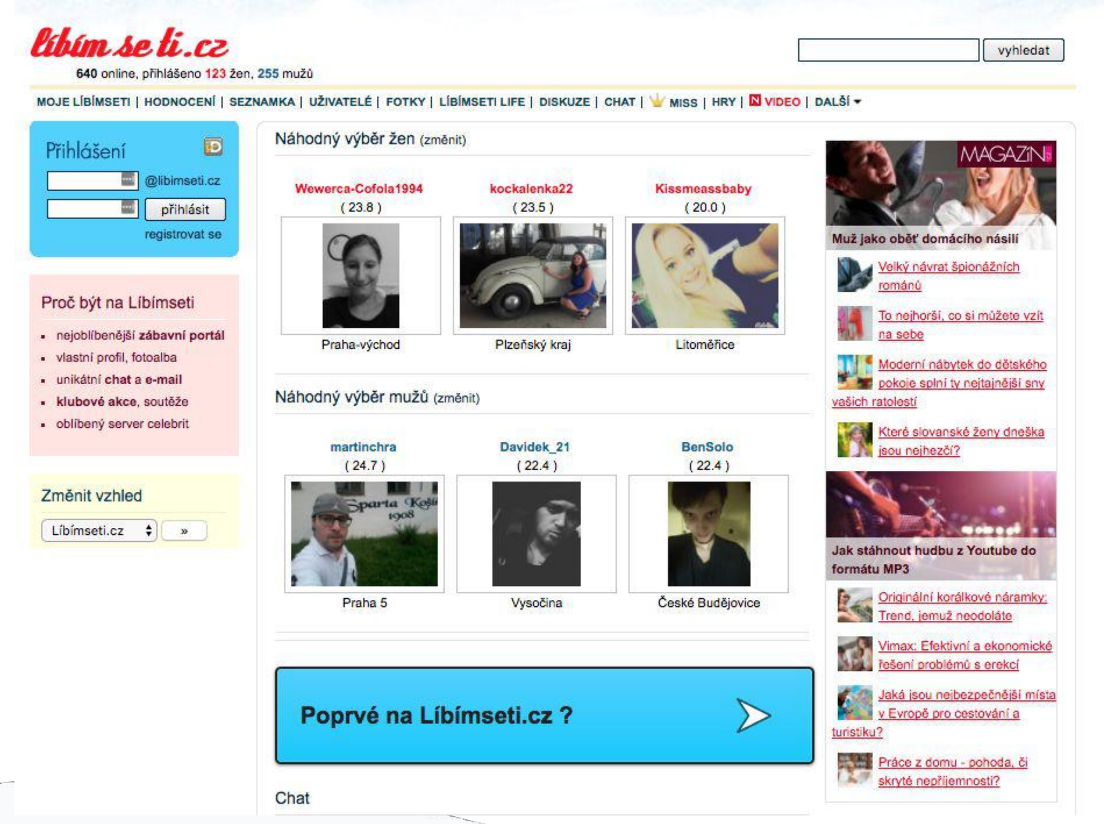
$$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2)$$

+ Netflix Prize (2009)
+ Solve via SVD (ALS or SGD)
+ Regression problem

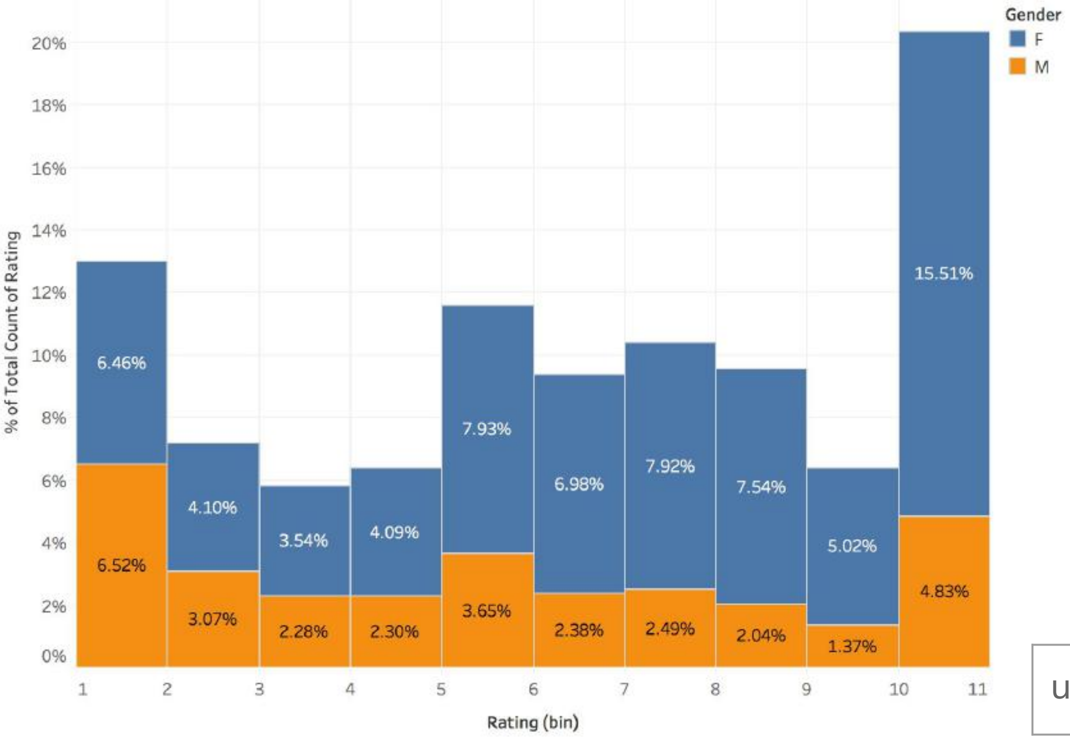
Finding Love with Numbers

datatonic

Online Dating Dataset - LibimSeTi



Online Dating Dataset - LibimSeTi



- http://www.libimseti.cz/
- 2005
- 17,359,346 ratings
- 135,359 users
- Ratings: 1-10

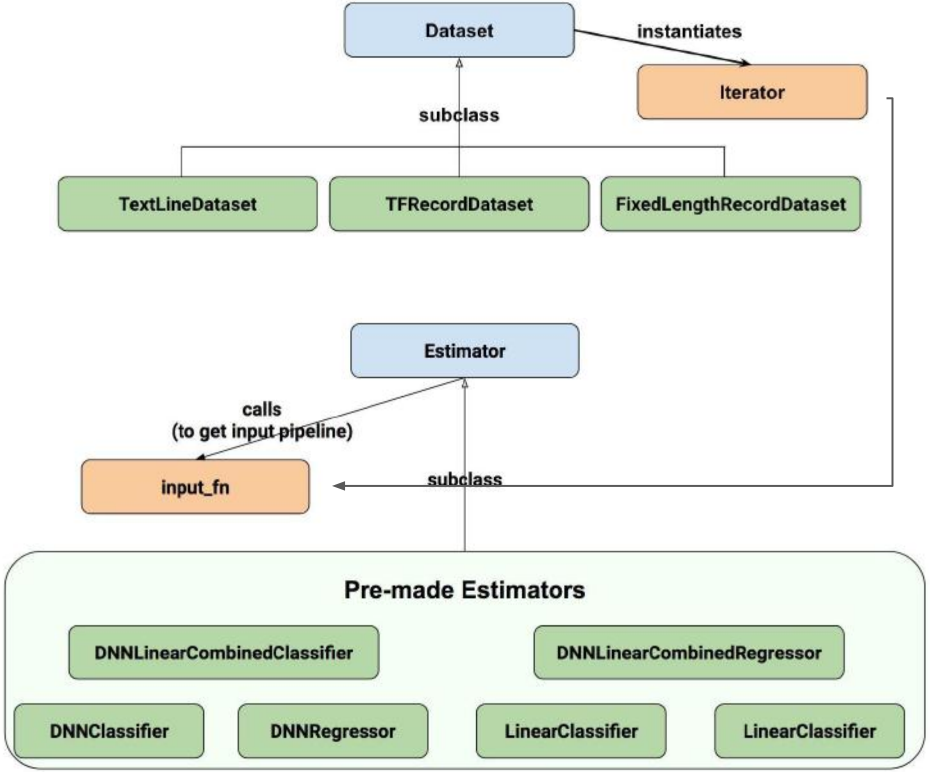
- Female (%): 69
- Male (%): 31
- Mean(rating): 5.9
- Std(rating): 3.1

userId	profileId	rating	gender

Tensorflow: High Level APIs

Dataset:

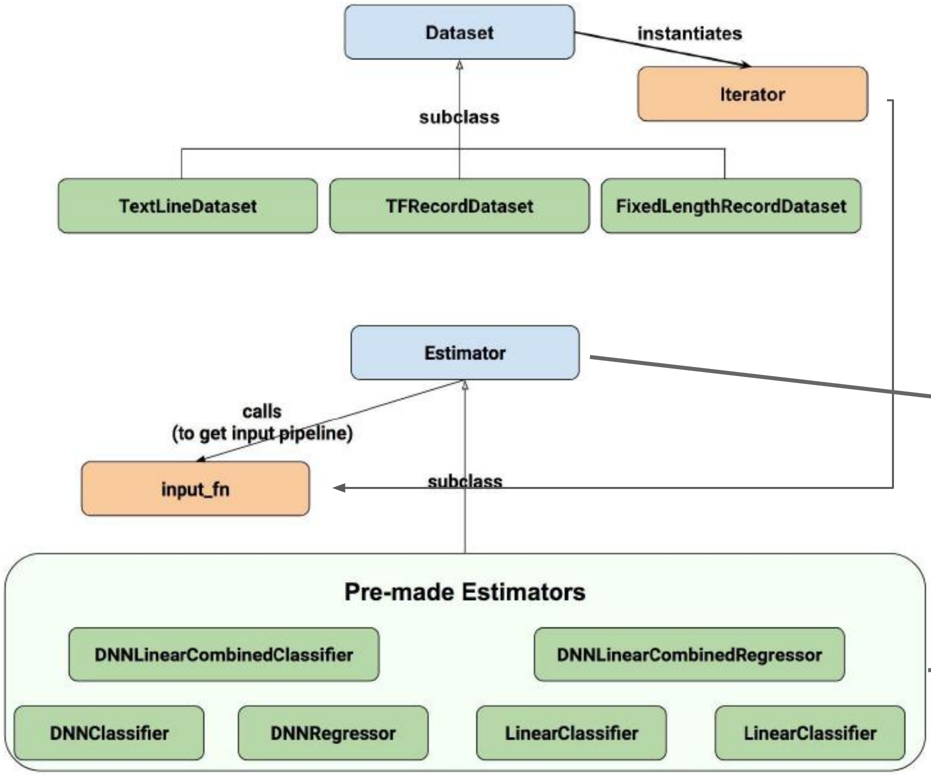
Estimator:



Tensorflow: High Level APIs

Dataset:

Estimator:

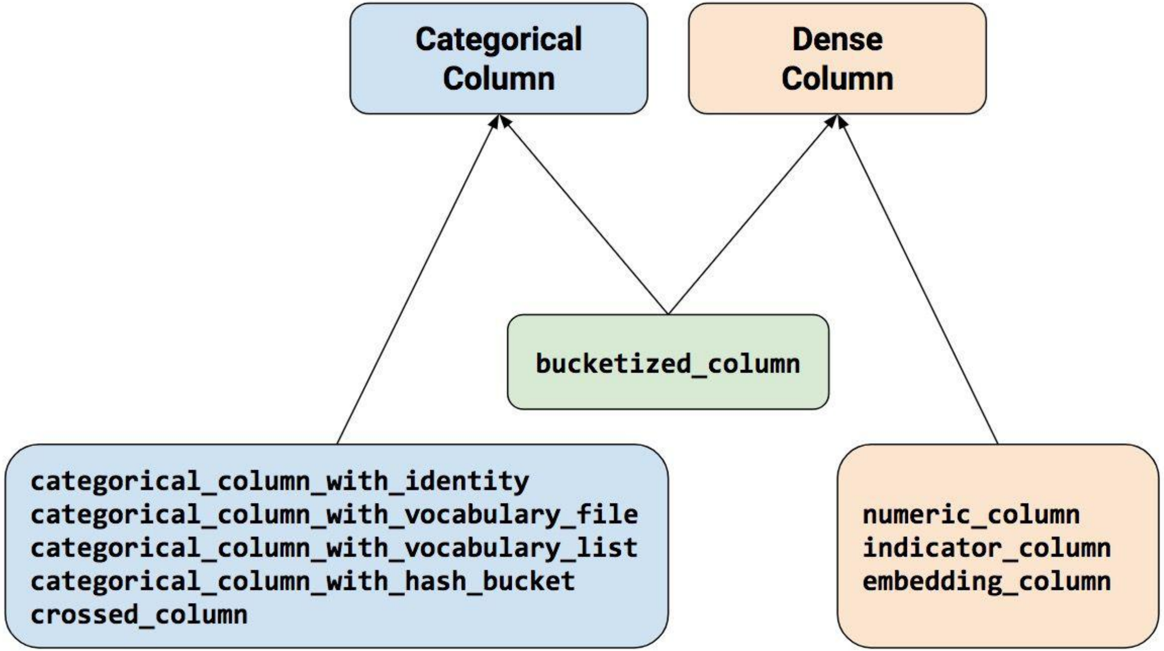


Easy distributed training (Cloud ML Engine, Kubeflow)

Boosted Trees
GMM
KMeans
AR Models

Tensorflow: High Level APIs

Feature Columns:



Dataset API (tf.data)

```
def input_fn(data_file, num_epochs=None, shuffle=True, batch_size=512, skip_header_lines=1):
    """Generate an input function for the Estimator."""
    def parse_csv(value):
        print('Parsing', data_file)
        columns = tf.decode_csv(value, record_defaults=CSV_COLUMN_DEFAULTS)
        features = dict(zip(COLUMNS, columns))
        labels = features.pop(LABEL_COLUMN)
        return features, labels

    # Extract lines from input files using the Dataset API.
    dataset = tf.data.TextLineDataset(data_file).skip(skip_header_lines)
    if shuffle:
        dataset = dataset.shuffle(buffer_size=10000)

    dataset = dataset.apply(tf.contrib.data.map_and_batch(parse_csv, batch_size, num_parallel_batches=1))
    dataset = dataset.cache()
    dataset = dataset.repeat(num_epochs)
    dataset = dataset.prefetch(1)

    iterator = dataset.make_one_shot_iterator()
    features, labels = iterator.get_next()

    return features, labels
```

MF Model

```
userid = tf.feature_column.categorical_column_with_hash_bucket("userid", hash_bucket_size=10000)
profileid = tf.feature_column.categorical_column_with_hash_bucket("profileid", hash_bucket_size=10000)
columns = [userid, profileid]
mf_feature_columns = [tf.feature_column.embedding_column(x, dimension=10) for x in columns]
mf_bias_columns = [tf.feature_column.embedding_column(x, dimension=1) for x in columns]

tensors = tf.feature_column.input_layer(features, mf_feature_columns)
biases = tf.feature_column.input_layer(features, mf_bias_columns)

userid, profileid = tf.split(tensors, 2, axis=1)
bias_userid, bias_profileid = tf.split(biases, 2, axis=1)
```

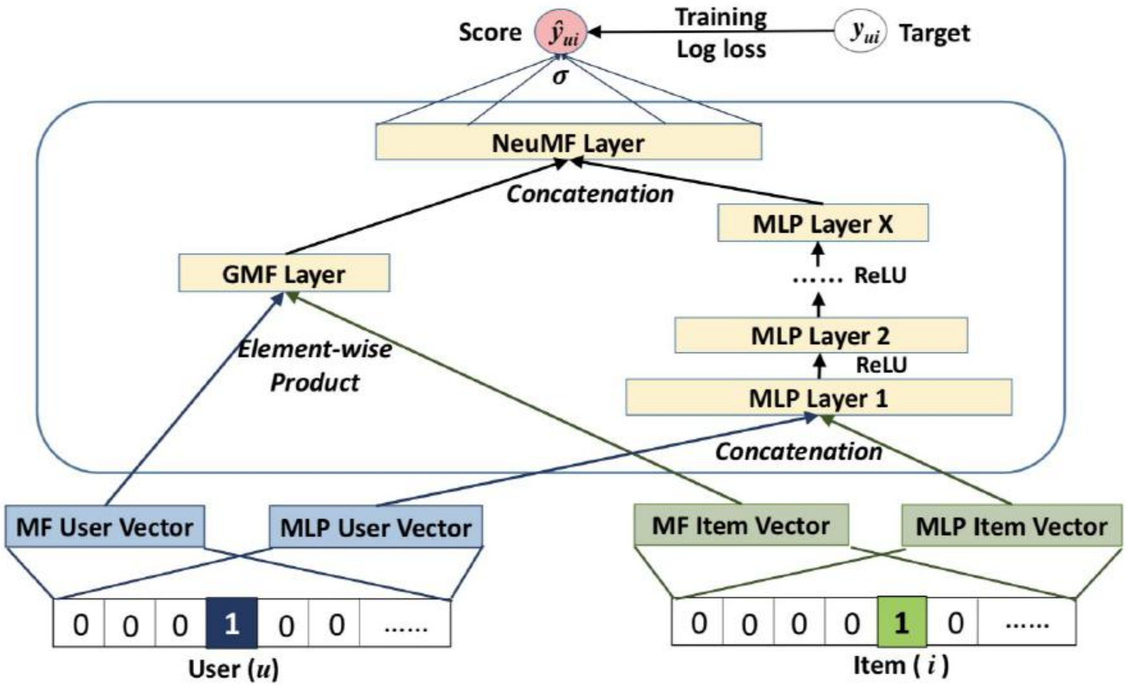
```
with tf.device(params['device']):
    # Calculate dot product
    model = tf.reduce_sum(tf.multiply(userid, profileid), 1)
    # Add biases
    model = tf.add(model, tf.squeeze(bias_userid, axis=1))
    model = tf.add(model, tf.squeeze(bias_profileid, axis=1))

    # Add regularization
    l2_reg = tf.contrib.layers.apply_regularization(
            tf.contrib.layers.l2_regularizer(params['l2_beta_MF'], scope="l2_reg"), weights_list=[userid, profileid])

    # Calculate loss using mean squared error
    loss = tf.losses.mean_squared_error(labels, model)
    loss = tf.add(loss, l2_reg)
```

Estimator API (tf.estimator)

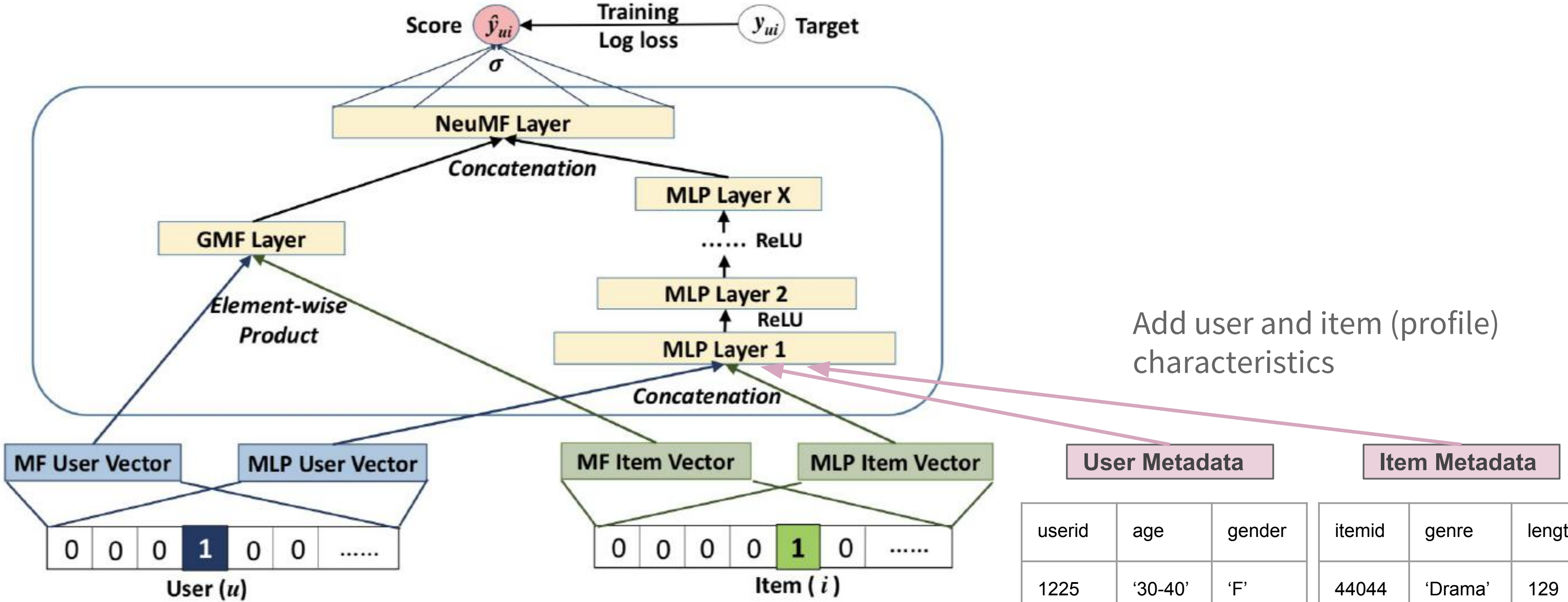
```
# Calculate metrics
eval_metric_ops = {
  "rmse":
      tf.metrics.root_mean_squared_error(labels, model),
  "mae":
      tf.metrics.mean_absolute_error(labels, model),
}

train_op = tf.contrib.layers.optimize_loss(
    loss=loss, global_step=tf.train.get_global_step(),
    learning_rate=0.001, optimizer='Adam')

model_fn = tf.estimator.EstimatorSpec(mode=mode, predictions=predictions_dict, loss=loss,
    eval_metric_ops=eval_metric_ops, train_op=train_op)

model = tf.estimator.Estimator(model_fn=model_fn, params=model_params, config=run_config, model_dir=model_dir)
```

Going Deeper - Beyond MF

Neural Collaborative Filtering (He et al.)



Going Deeper - Beyond MF



userid	age	gender
1225	‘30-40’	‘F’

itemid	genre	length
44044	‘Drama’	129

Results - LibimSeTi

	MF	MLP	MF + MLP	Research [1]
RMSE	2.137	2.112	2.071	2.077
MAE	1.552	1.541	1.432	1.410

Training details:

- 40 epochs
- MLP: 4 layers (256 units pyramid)
- Adam optimiser
- Results calculated on held out test set (5 rating per user)
- No tuning of hyperparameters

[1] Trust-Based Recommendation: an Empirical Analysis, O'Doherty, Jouili, Van Roy (2008)

How can we do better?

datatonic

Better/More Data

Better/More Data

Single Customer View

Typical information available to Retailers & CPG companies

Personal Details	Demographics	Rule-Based Segmentation
Purchase History	Online Interactions	Support Interactions

Pro-Active Machine Learning Model Output

Propensity Scores	Dynamic Segmentation	Churn Prediction
Recommendations	Lifetime Value	Sentiment Scores

Analytics & Forecasting

Personalised Web Experience

Personalised Advertising

Customer Interactions

Dynamic Pricing

Better Loss Functions

Better Loss functions

+ Implicit feedback (Hu, Koren, Volinsky 2008):

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

+ Logistic Matrix Factorisation (Johnson, Spotify, 2014):

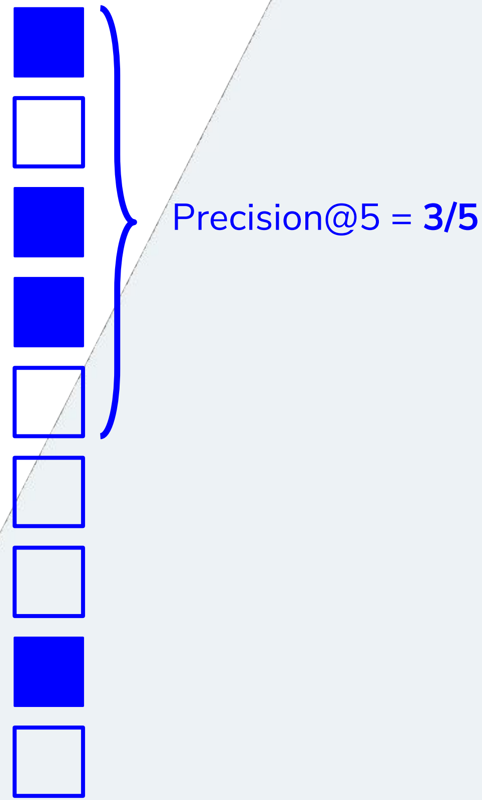
$$p(l_{ui} \mid x_u, y_i, \beta_i, \beta_j) = \frac{\exp(x_i y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)}$$

+ Use ranking loss (or pairwise loss functions)

Improved Metrics

Precision@k

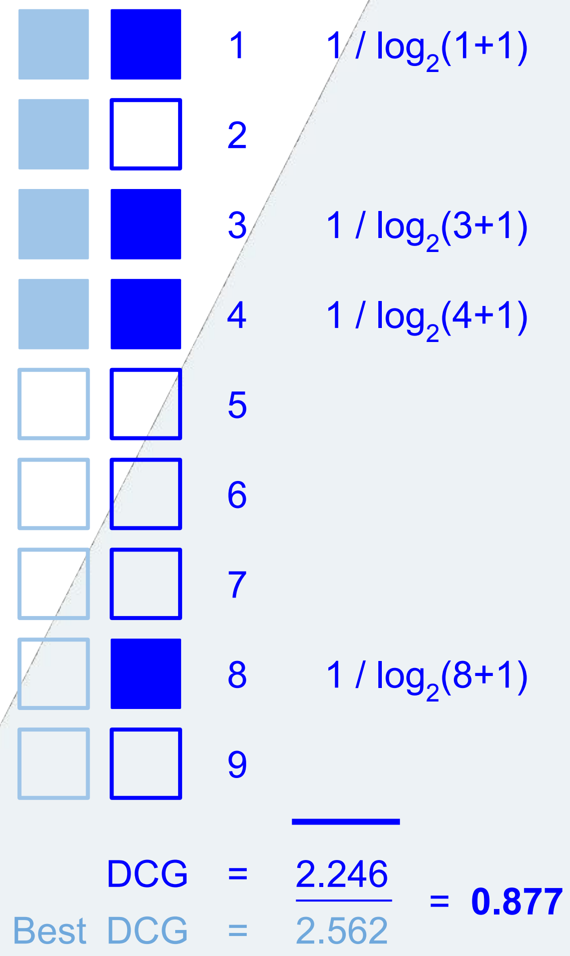
Of the top **k** recommendations, how many were relevant (interacted with)?



(Normalized) Discounted Cumulative Gain

DCG given by formula

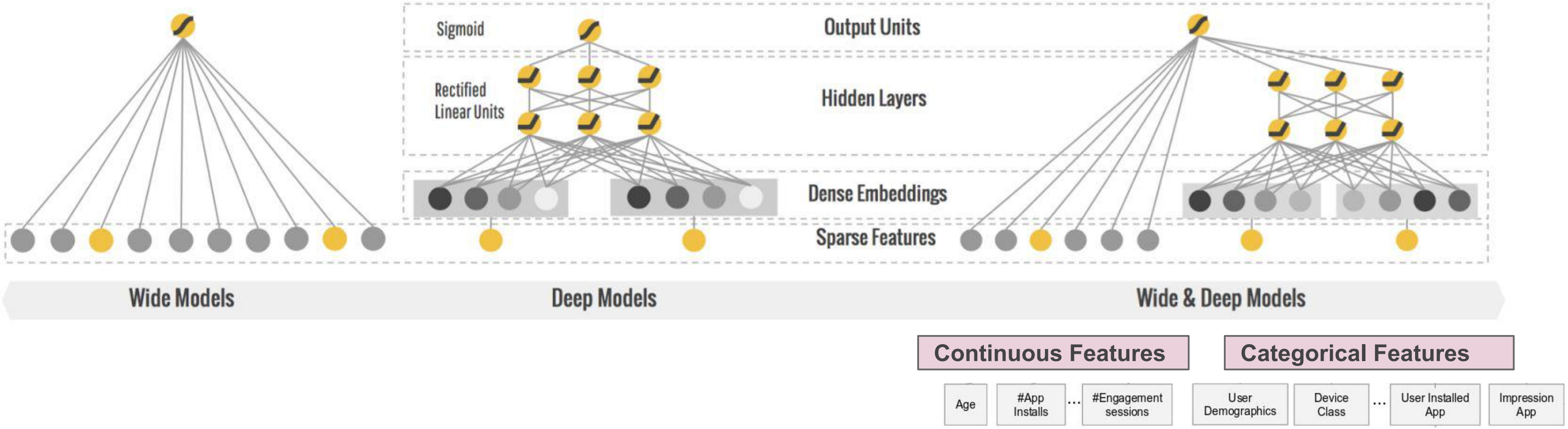
$$\sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$



Go even deeper and embed all the things!

Deep Recommender Systems - Advances

Wide & Deep model (Cheng et al., 2016)



Results - Real Client Data

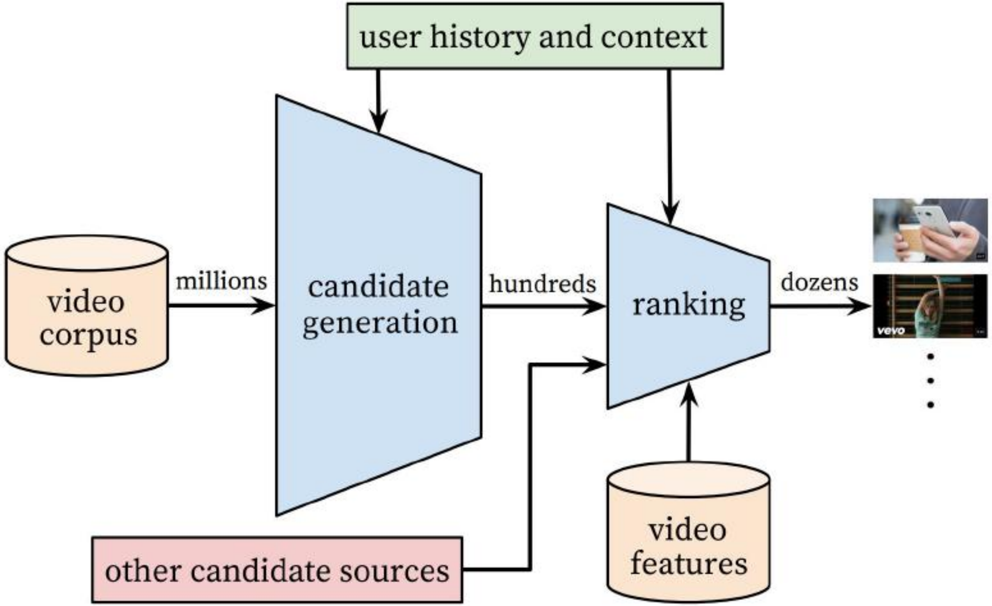
	SVD (MF)	MF - Implicit Feedback model	Wide & Deep
P@5	0.33	0.51	0.79
NDCG@5	0.18	0.30	0.37

Training details:

+ Feature columns include user demographics to complement the lack of interactions of cold users
+ 100 epochs
+ Adam optimiser
+ Results calculated on held out test set (up to 5 ratings per user)
+ Tuning of the dimension size of the embedding vector of the deep part

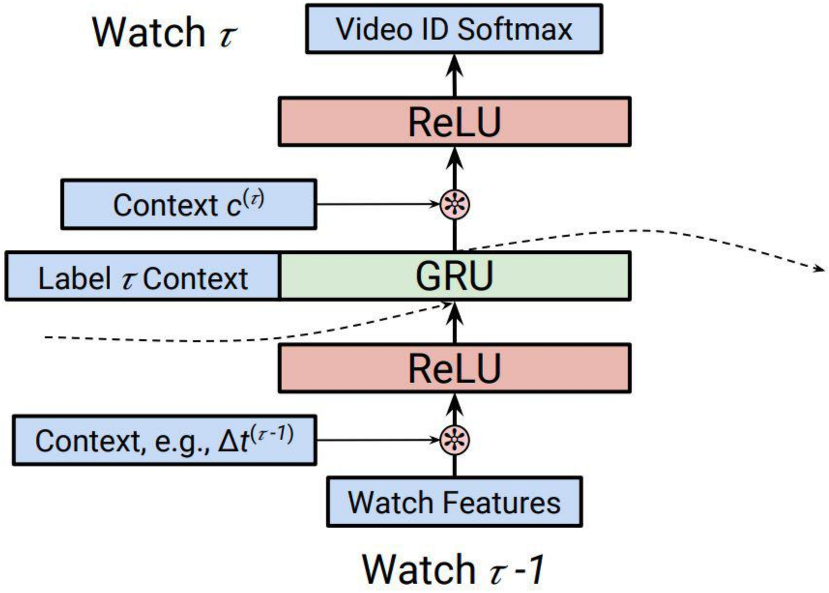
Deep Recommender Systems - Advances

Deep Neural Networks for YouTube Recommendations (Covington, Adams, Sargi, 2016)



Deep Recommender Systems - Advances

Latent Cross: Making Use of Context in Recurrent Recommender Systems (Alex Beutel, Paul Covington, Sagar Jain et al. , 2018)



Takeaways

Takeaways

+ Tensorflow can do more than vision or translation
+ High level APIs make model building and training painless
+ Custom algorithms and specific loss functions are easily implemented
+ Deep Recommender systems work well on real data
+ Embeddings and hidden layers allow for many ways to improve a recommender system

Thank you.

Blog.datatonic.com

facebook.com/datatonic

linkedin.com/company/datatonic

twitter.com/teamdatatonic