Coupon Purchase Prediction

Team BruteForce - November 6, 2021
CSCE 5300.002

Team Members

Prathima Bommannagari

Ganesh Bommisetty

Shasidhar Jampana

Akhil Kata

Harshada Pol

Siva Kumar Sri Sai Pulavarthy

Dan Waters

Agenda

- **Introduction**
    - Business Understanding
    - Problem Class
    - Assumptions
    - Evaluation Criteria
- **Data**
    - Data Source
    - Preprocessing
    - Exploratory Data Analysis
- **Solution**
    - Baseline
    - Logistic Regression (predict based on browsing history)
    - Cosine similarity (compare purchased coupons with test set coupons)
    - Decision Forest and Gradient Boosted Trees
    - Comparisons
- **Results & Conclusion**
- **Future Enhancements**
- **Q&A**

Introduction

Business Understanding

Recruit Ponpare (a website like Groupon) **partners with businesses** to sell discounted services to end customers via prepaid **coupons**.

Example: *$25 birthday cake for $10 at BruteForce Bakery*

Local business

Uses coupons as a marketing tactic to drive business with pre-paid discount sales

Coupon broker

Lists coupons from local businesses to a captive audience of coupon buyers for a fee

End users

Everyday people engage with local businesses, thanks to the coupon marketplace

Project objective:

Predict which 10 coupons a user is most likely to purchase from 6/24/2012 - 6/30/2012

Problem Class

- Recommender systems are usually **supervised learning** tasks:
  - The user has entered information about what they like (purchases, reviews, ratings, etc), giving us **labeled training data**. We provide one such approach.
- There are also **unsupervised** approaches...
  - K-means clustering to find similar coupons to ones already purchased.
- And **algorithmic** approaches...
  - such as cosine similarity evaluation and collaborative filtering.

In our project, we consider three general approaches:

- supervised learning with hybrid collaborative filtering (logistic regression)
- decision forests & gradient boosted trees
- hybrid collaborative filtering using cosine similarity

Assumptions

- Working within the scope of the Recruit Ponpare Kaggle competition
  - Data, problem framing are provided
- The data we have is **all that is available** (no other datasets to include)
- **Fixed timeframe** for browsing & purchases for train and test periods

Understanding the Dataset

Dataset Overview

Our data comes from a Kaggle competition called "Coupon Purchase Prediction," which contains data from Japanese joint coupon website Ponpare.jp.

The data set contains:

- **One year** of historical coupon transaction data (July 2011 to June 2012)
- **22,873 users**
- **Coupon details** (genre, location, % discount, price)
- **Coupon purchase history**
- **Browsing history**
- **Prefecture locations** (names and lat/lon coordinates)

Test sets for the week of 24 June 2012 - 30 June 2012 are held-out.

Dataset Challenges

- Some columns have values in Japanese, requiring a translation step in the preprocessing pipeline
- Many NaN values - some are meaningful, others not
  - For example, NaN in a user’s withdrawal date means they are active on the website
  - But NaN for Prefecture means the user has not completed their profile
- Several tables to join together

Tables

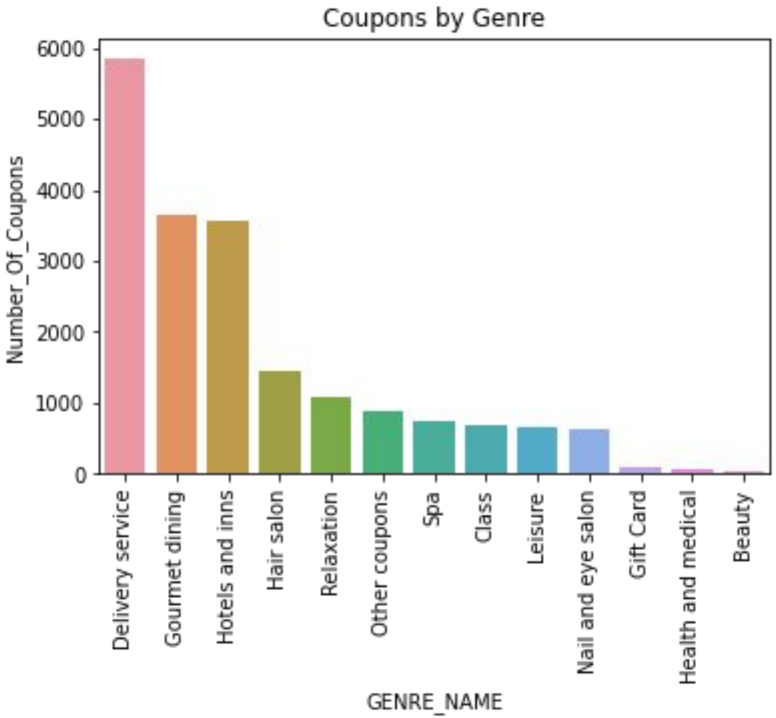
We have multiple tables to consider in our project:

- User List
- Coupon Listings (training & test data sets)
- Coupon Transaction History (training only)
- Coupon Areas (training & test data sets)
- Coupon Visits (training only)
- Prefecture Locations

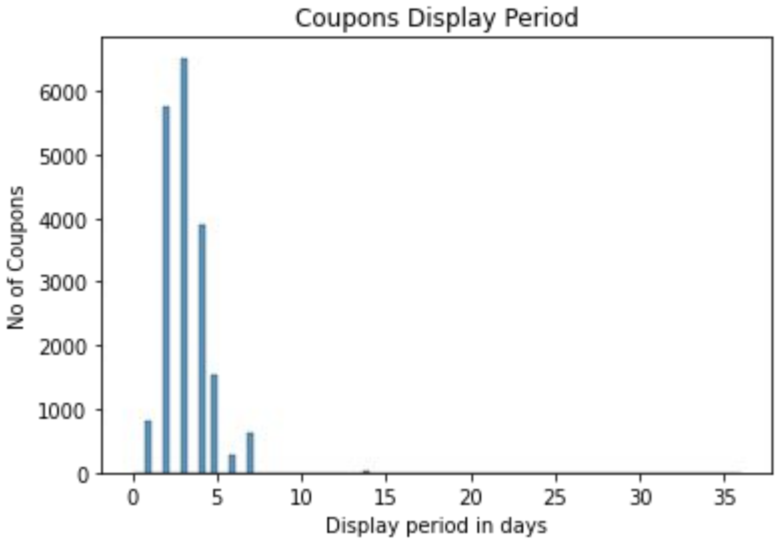
An entity-relationship diagram is available in the appendix.

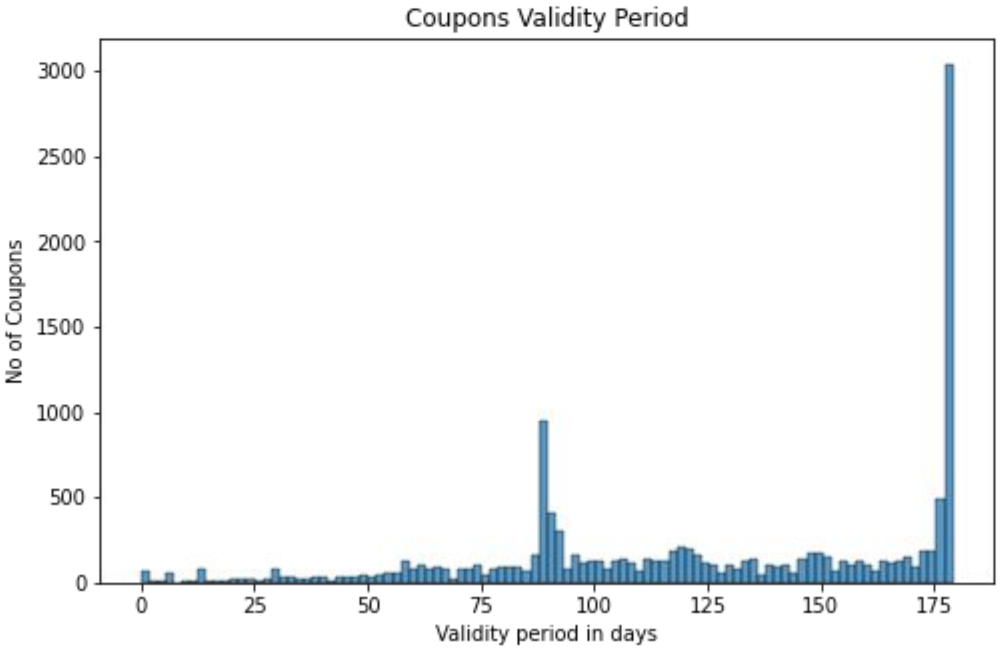
Exploratory Data Analysis

Analysis: Coupon List (Genre & Discount Rate)



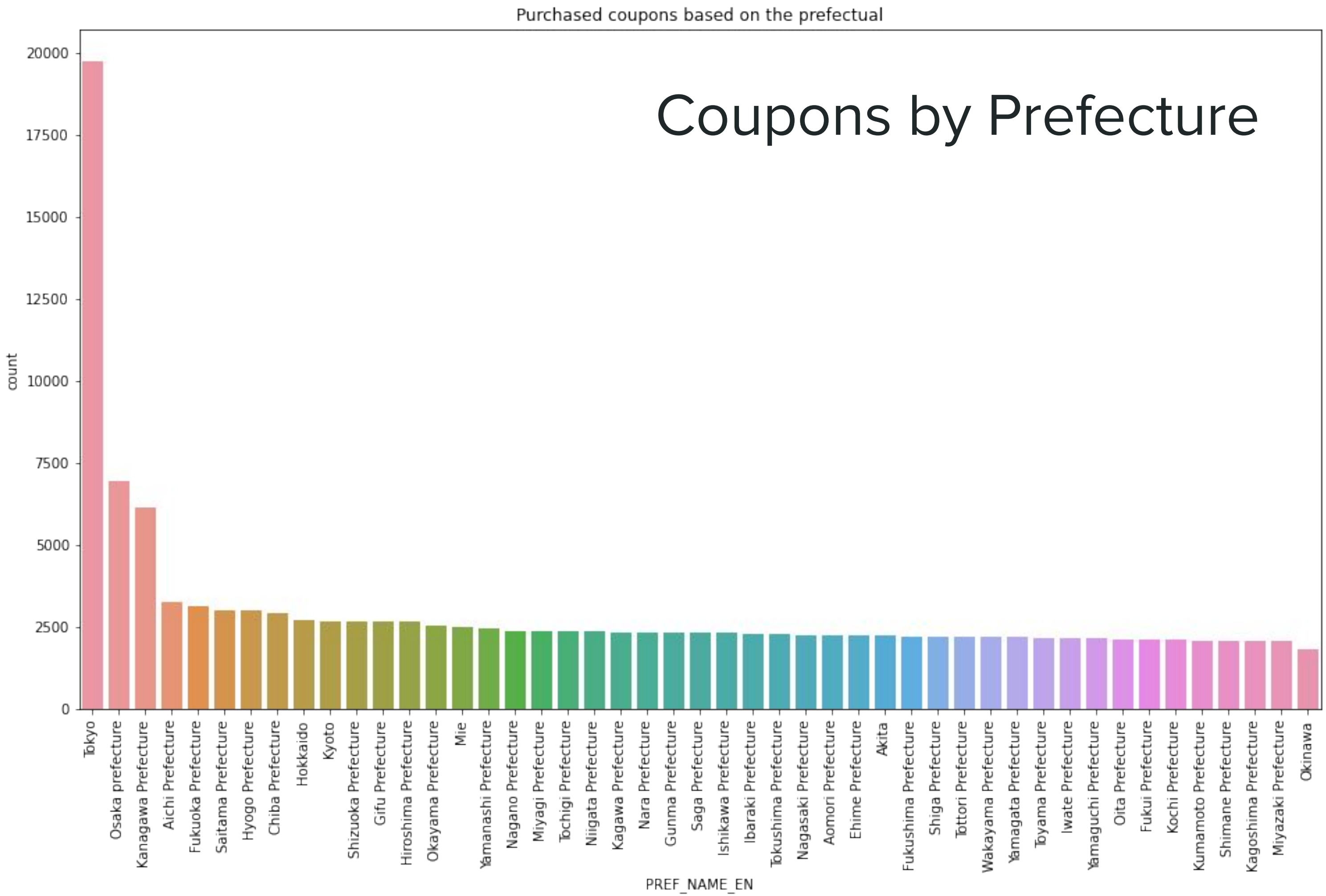
Analysis: Coupon List (Display & Validity Periods)





Coupons by Small Area

Coupons by Prefecture

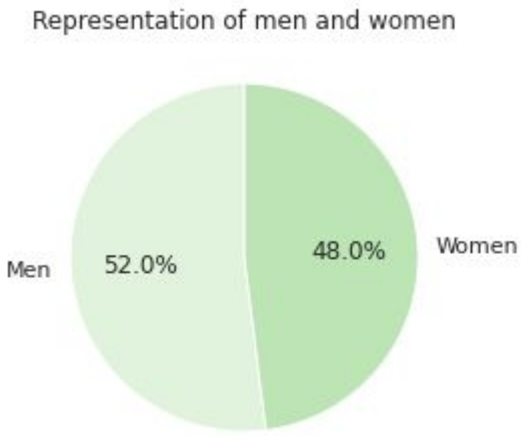


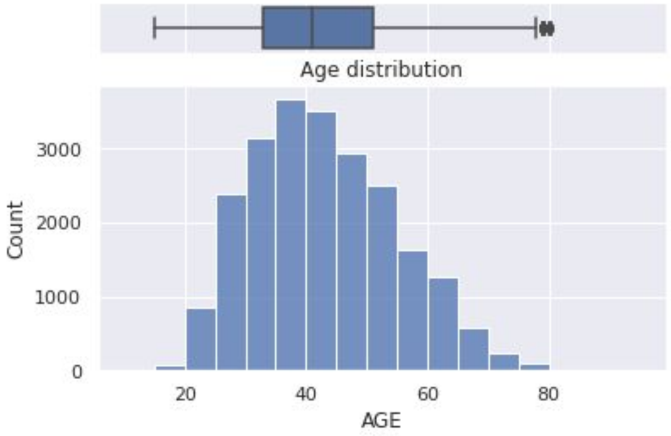
Analysis: User List

22,783 *records*

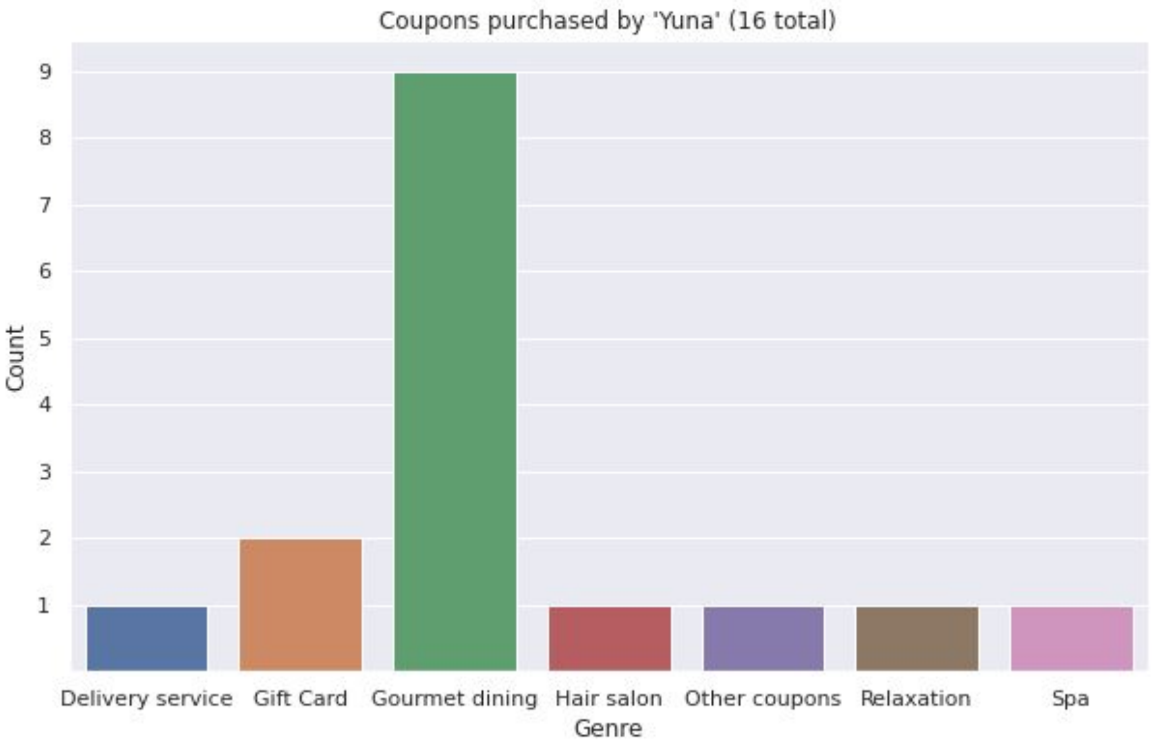
Columns	
REG_DATE	Registration date (datetime)
SEX_ID	M or F (categorical)
AGE	numeric
WITHDRAW_DATE	Date of unregistration (if applicable)
PREF_NAME	Prefecture in Japan where user is located (*Japanese*)
USER_ID_hash	Unique user ID

= potential demographic targeting feature





Individual User Persona ("Yuna")



About "Yuna"

- **Female, age 32**
- Lives in **Tokyo**
- Really likes **food**
- Only bought one coupon outside of Tokyo

Solutions

Evaluation Criteria - Mean Average Precision @ 10

Mean Average Precision (k=10) is the evaluation metric Kaggle uses to judge entries on its test set for this competition, so we will be optimizing MAP@10. **The best public MAP score is 0.01268**.

mAP is the **mean** of the **average precisions** of the top *k* classes (in this case, top 10 coupons predicted for a user).

Other evaluation metrics may be used during training monitoring.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

Glossary

Q	number of classes, equal to ***k*** for us
q	individual class being averaged
AveP()	average precision for a single class

Source: "Mean average precision." Wikipedia, https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)#Mean_average_precision

Baseline Solution

Random Sampling (MAP@10 result = `0.00051`)

```
# create random baseline
user_list = df_users['USER_ID_hash']

rows = []
for u in user_list:
  coupon_list = df_c_list_test.sample(n=10, replace=False)['COUPON_ID_hash']
  coupon_list_str = ' '.join(coupon_list)

  row = {'USER_ID_hash': u, 'PURCHASED_COUPONS': coupon_list_str}
  rows.append(row)

df_pred = pd.DataFrame.from_dict(rows)
df_pred.to_csv('sample_submission.csv', header=True, index=False)
```

Algorithm:

For each user, randomly select 10 coupons from the test set and recommend them.

Surely we can do better than this!

Solution #1: Logistic Regression on Browsing Data

Step 1: Join and preprocess raw, **anonymized** training data

a. Browsing data + user data, joined on `USER_ID_hash`
b. Join the above to coupon details on `COUPON_ID_hash`
c. Create age ranges instead of continuous ages (for user-based collaborative filtering)
d. Convert **`sex_id`** from **`m/f`** to **`0/1`**

Resulting training fields:

`age_group`	`sex`	`discount_rate`	`discount_price`	`user_prefecture`	`coupon_prefecture`	`genre`	`capsule_text`	`purchase_flag`

training features *target*

Solution #1: Logistic Regression on Browsing Data

Step 2: **One-hot encode** the categoricals (resulting in 139 input columns)

Step 3: Build and train the model - simple dense NN with one hidden layer and sigmoid activation

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 16)                2240
_________________________________________________________________
dropout_5 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 17
=================================================================
Total params: 2,257
Trainable params: 2,257
Non-trainable params: 0
_________________________________________________________________
```

Hardware-accelerated training time: ~55 min
Inference time (mapping coupons to users): ~1 hour

Validation precision & recall: 0.0 - What?
Kaggle MAP@10 score on test set: 0.00028

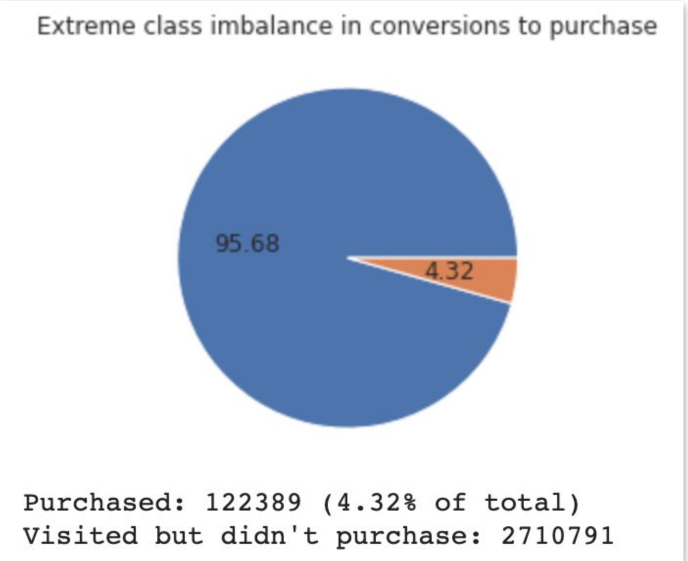
This is worse than random sampling... why?

Solution #1: Logistic Regression on Browsing Data

For browsing data, the positive class ('the user bought something') is **extremely** underrepresented!

We will take this same approach, but:

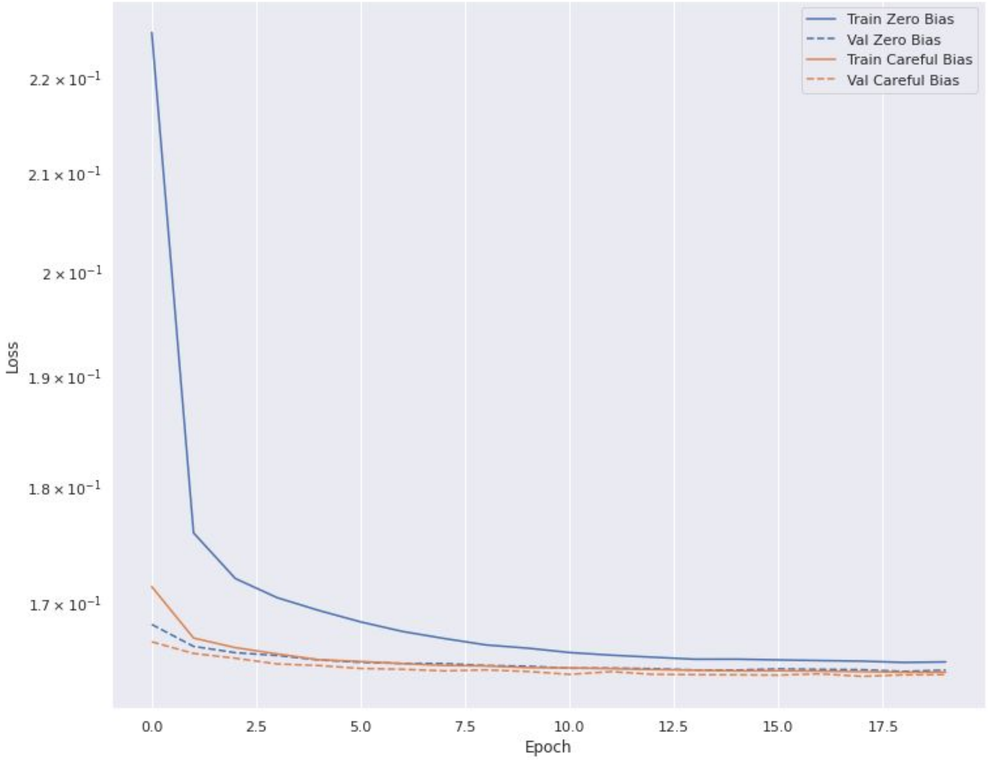
- Introduce feature scaling (fit on training set)
- One-hot encode additional categoricals (146 cols)
- Use stratified sampling in `train_test_split`
- Initialize the bias to `np.log([pos/neg])`
- Implement **early stopping**
- Compute the **class weights** so that the model pays more attention to the *purchases*



For a comprehensive tutorial on dealing with imbalanced classes, see:
https://www.tensorflow.org/tutorials/structured_data/imbalanced_data

Solution #1: Logistic Regression on Browsing Data

Setting initial bias of the network helped the network learn faster and reduce loss.



Solution #1: Logistic Regression on Browsing Data

Results

- MAP@10 = **0.00061**
- Baseline = **0.00051**

Slight improvement - but nothing groundbreaking

Solution #2: Cosine Similarity with Collaborative Filtering

This method computes the **cosine similarity** between coupons the user has bought with coupons in the test set. This solution is **memory-based,** not **model-based**

Comparison Fields: `['discount_rate', 'discount_price', 'capsule_text', 'genre', 'large_area', 'small_area', 'coupon_prefecture', 'age_group', 'sex', 'user_prefecture']`

Merge user data with test coupons	One-hot encoding	Return 10 max similarity
For every coupon a user has purchased, create a comparison vector containing that coupon's data and the user's metadata (age_group, sex, prefecture). Also compute a matrix of all 310 test coupons.	One-hot encode all the categorical variables in the "purchased" coupon and the test coupon matrix.	Compute cosine similarities between all 310 coupons and all n coupons a user has purchased. Select the 10 coupons with the highest similarity score to any coupon the user bought in the past.

Solution #2: Cosine Similarity with Collaborative Filtering

Results (scored by Kaggle on held-out test data)

- Cosine Similarity Hybrid CF = **0.00269** **Wow!** *(Top 73%)*
- Class-Weighted Logistic Regression = 0.00061
- Baseline (Random Sampling) = 0.00051
- Logistic Regression (Unweighted) = 0.00028

Cosine similarity using a user's actual purchases is an **order of magnitude better** than trying to use browsing history to predict purchases.

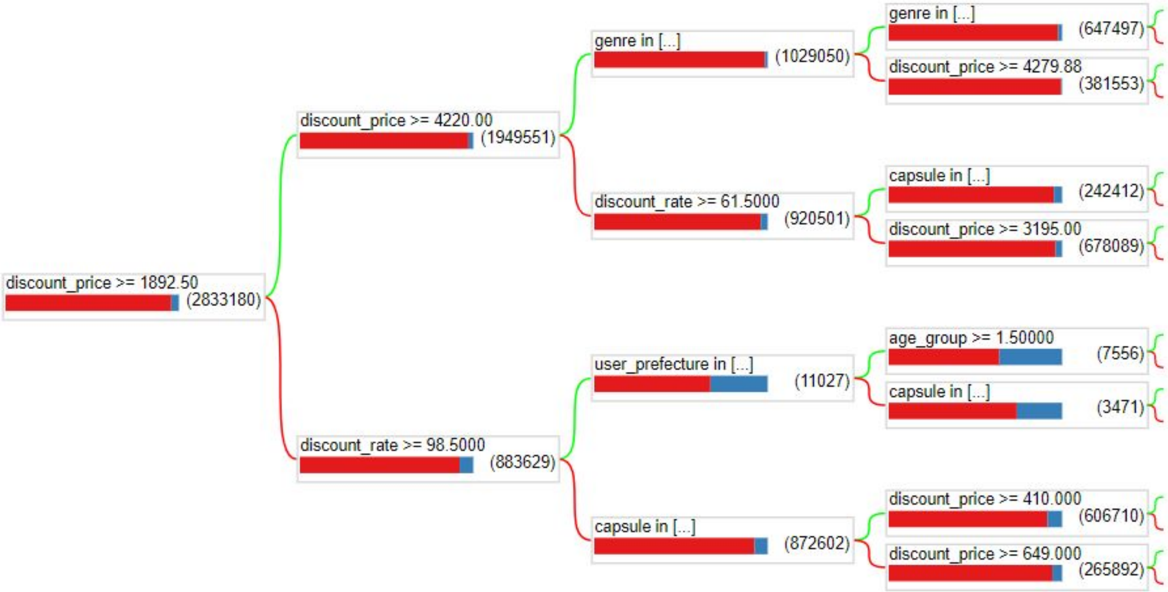
However - if a user has never purchased anything, this algorithm predicts nothing for them - maybe we can backfill it with predictions based on browser history!

Solution #3a: Decision Forest (TFDF)

Trained on the same unbalanced browsing data as logistic regression

MAP@10: 0.00047

Baseline: 0.00051



Decision trees are weak classifiers and
are usually outperformed by gradient boosted trees.[1]

1. Gradient Boosting (Wikipedia) https://en.wikipedia.org/wiki/Gradient_boosting

Solution #3b: Gradient Boosted Trees (TFDF)

Trained on the same unbalanced browsing data as logistic regression

MAP@10: **0.00263 (!)** *Similar performance as cosine similarity on purchase history!!*

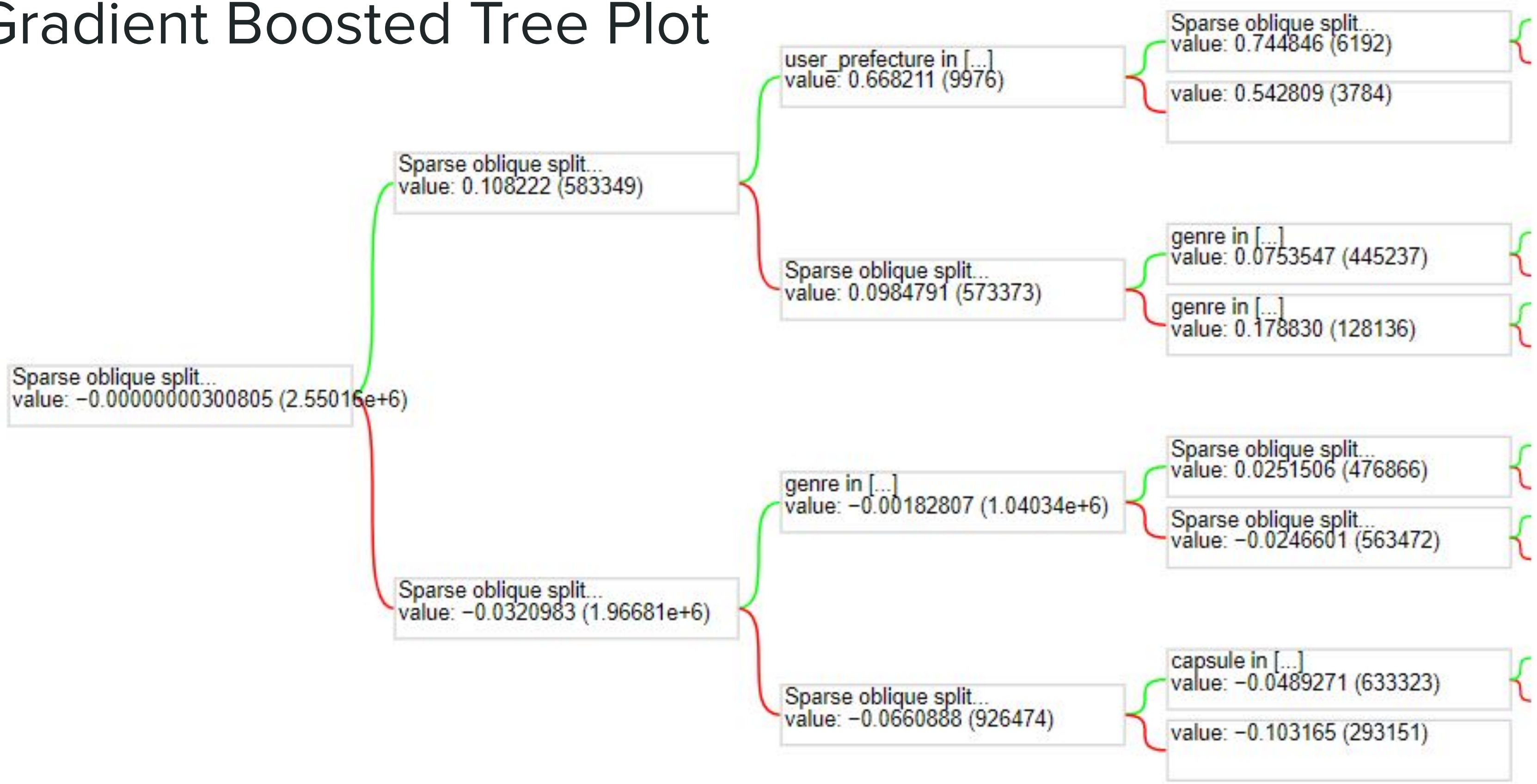
Baseline: **0.00051**

```
ds_train_set = tfdf.keras.pd_dataframe_to_tf_dataset(df_train, label='purchased')

model = tfdf.keras.GradientBoostedTreesModel(
     num_trees=500,
     growing_strategy='BEST_FIRST_GLOBAL',
     max_depth=8,
     split_axis='SPARSE_OBLIQUE')

model.fit(ds_train_set)
```

Gradient Boosted Tree Plot



Results

Solution Comparison Matrix

*Baseline MAP@10 score on Random Sampled submission: **0.00051***

	Logistic Regression	Logistic Regression with Class Weights	Cosine Similarity	Decision Forest	Gradient Boosted Decision Trees
Details	No Class Imbalance Correction	Corrected for class imbalance	no backfilling	Using TensorFlow Decision Forests (Random Forest and CART)	TF-DF Gradient Boosted Trees with hyperparameters: 500 trees Best First Global growing strategy Max depth 8 Sparse Oblique split axis
MAP@10 Kaggle Public	0.00028	0.00061	0.00269	0.00047	0.00263
Pros	Easy to configure Prediction column exists for training	Easy to configure Optimized for class imbalance	Best performance thus far (top 73%)	Easy to code No preprocessing necessary	Easy to code No preprocessing necessary Great performance
Cons	It’s terrible Training and inference time > 2hr	Still not very accurate compared to other solutions	Extremely long inference time **Computationally expensive** No results for users who don’t buy anything	Worse than baseline (needs HP tuning?)	Maybe *too* easy... :)

New Work Since Our Presentation

Baseline MAP@10 score on Random Sampled submission: ***`0.00051`***

We took the Cosine Similarity submission and found **91 users** without assigned coupons. This is because those users had no purchase history to predict from.

Then, we used the Gradient Boosted Trees predictions for those users (based on browsing history) and assigned those coupons to them for the submission.

This resulted in the best score of all attempts.

Result: MAP@10 = **0.00283** (public), **0.00336** (private)

Conclusions

Initial conclusions:

- Browsing history has less-than-expected correlation with purchasing habits
- Past purchasing habits are better indicators of future purchases
- **Cosine Similarity** on purchase data **far outperforms Logistic Regression** (as a binary classifier) on browsing data

HOWEVER:

- **Gradient Boosted Trees** performs at the same level as cosine similarity, though they are trained on browsing data. This merits further investigation.

Future Enhancements

To be completed by project deadline:

- Include cosine similarity with backfilling based on browsing history (maximize MAP to predict for users without purchase history) (Completed)

Aspirational goals:

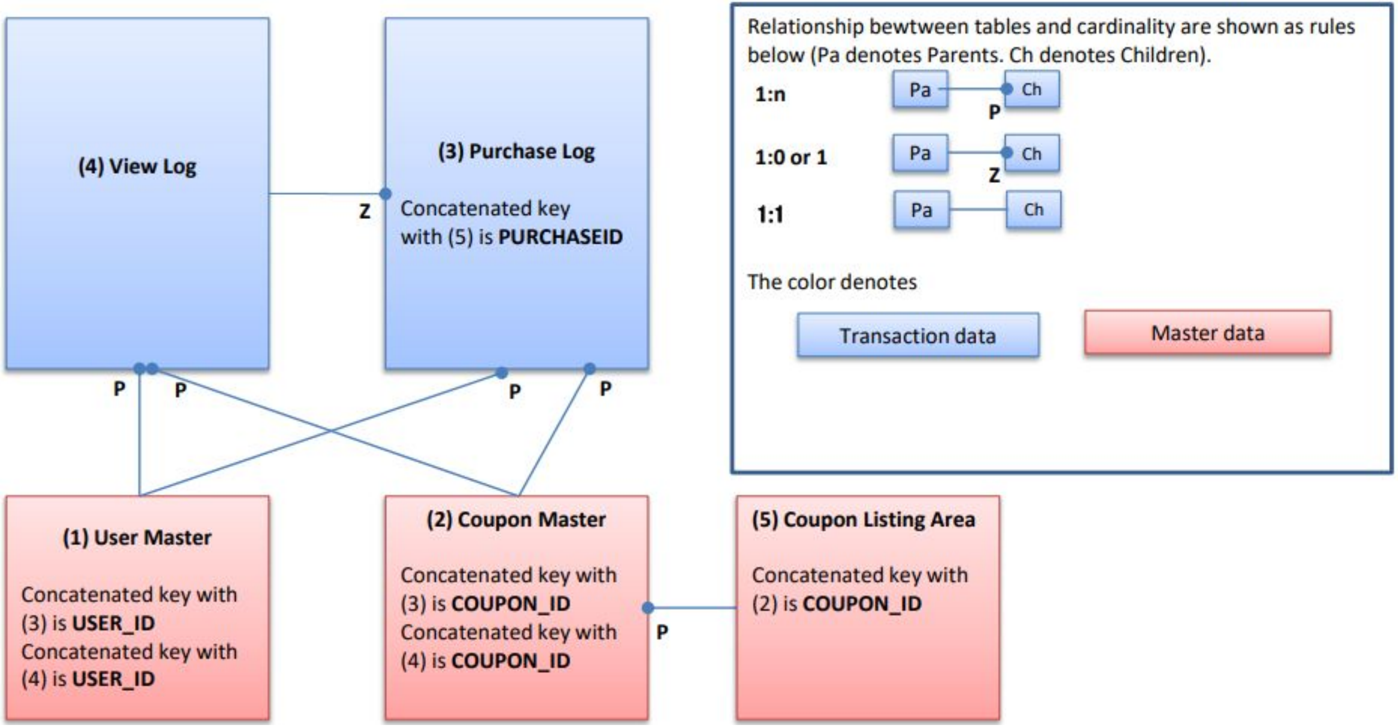
- If a user has not browsed anything, include coupons for similar demographic
- Tune hyperparameters for gradient boosting
- Experiment with different dataset selection techniques
- EDA and training for coupon area table (more location-aware predictions)
- Aim for public MAP@10 score on test set > `0.008` (top 100 / top 10%)
- Create time machine to back in time and win $50,000

Thank You!

Q&A

Appendix

Entity Relationship Diagram



Source: Provided with the Kaggle competition documentation