# RecSys Assignment 2 Ideas

## Github Repo

<https://github.com/AwsManas/Assigment-2>

## Assignment 2 Slides

<https://cseweb.ucsd.edu/classes/fa22/cse258-a/slides/assignment2_fa22.pdf>

## Datasets

1. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#google_local>
2. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews>
3. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#twitch>
4. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data>
5. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#behance>
6. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#social_data>
7. <https://cseweb.ucsd.edu/~jmcauley/datasets.html#reddit>
8. [Citation Network Dataset | Kaggle](https://www.kaggle.com/datasets/mathurinache/citation-network-dataset) - Citations Dataset 12 GB
9. [UK Road Traffic Collision Dataset | Kaggle](https://www.kaggle.com/datasets/salmankhaliq22/road-traffic-collision-dataset) - Uk Roads Dataset
10. [Road Safety Data - data.gov.uk](https://www.data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data)
11. [UK Road Safety: Traffic Accidents and Vehicles | Kaggle](https://www.kaggle.com/datasets/tsiaras/uk-road-safety-accidents-and-vehicles)
12. [National Highway Traffic Safety Administration](https://cdan.nhtsa.gov/) / [NCSA and Other Data Sources](https://cdan.nhtsa.gov/Homepage/MotorVehicleCrashDataOverview.htm)
13. [Toward Road Safety Recommender Systems- Formal Concepts and Technical Basics.pdf](https://drive.google.com/file/d/1Qhuxv1vYY14fjC4IWMJ1tzEsQjsFbACl/view?usp=sharing)
14. [Using Machine Learning to Predict Car Accidents | by Eugenio Zuccarelli | Towards Data Science](https://towardsdatascience.com/using-machine-learning-to-predict-car-accidents-44664c79c942)
15. [Local Business (LocalBusiness) Structured Data | Google Search Central | Documentation](https://developers.google.com/search/docs/appearance/structured-data/local-business)

## Steps

1. Data Collection
2. Data pre-processing
3. Feature Extraction
4. Problem Formulation
5. Baseline Models
6. Proposed Model
7. Model Evaluations
8. Results
9. Literature Review / Related Work

## EDA

This step does the following:

1. Finds if there are columns with duplicate names and lists the columns.
2. Finds counts of rows in the dataframe. Computes the following counts for entire dataframe as well as each column:
   1. Total count with null values
   2. Total count without null values
   3. Count with null values and without duplicates
   4. Count without null values and without duplicates
   5. **Completeness**: Fraction of non-null values over total values
   6. **Distinctness**: Fraction of distinct non-null values over total non-null values
   7. **Uniqueness**: Fraction of unique values over the number of all non-null values of a column. Unique values occur exactly once. Example: [a, a, b] contains one unique value b, so uniqueness is 1/3.
   8. **Unique value ratio**: Fraction of unique values over the number of all distinct non-null values of a column. Unique values occur exactly once; distinct values occur at least once. Example: [a, a, b] contains one unique value b, and two distinct values a and b, so the unique value ratio is 1/2.
3. Check if the items in a column are case sensitive. Lists such columns.
4. Finds the possible primary keys (individual columns) in the dataframe.
5. Finds the mapping of data types to column names.
6. Finds the most frequent items (top-k along with their percentage of occurrences) in each column.
7. Finds the following stats for each column in the dataframe:
   1. Mean value
   2. Standard deviation
   3. Minimum value
   4. Maximum value
   5. Percentiles
   6. Entropy

For string type columns, the above computations are done on the length of the strings. For boolean type columns, the above computations are done after converting them into integer type columns. Array type and struct type columns are ignored.

1. Lists the columns with no cardinality, low cardinality (<100 unique values) and their distributions.
2. Lists the columns with high cardinality (>=100 unique values).
3. Finds all pairs of columns in the dataframe which are strongly correlated (both positive and negative) and which are not correlated.
4. Finds outliers and columns with outliers:
   1. **99-100-percentile-outliers**: The data contains 99-100-percentile outliers if the difference between the 99.999th percentile and 100th percentile exceeds a certain threshold.
   2. **z-score-outliers**: The data contains z-score-outliers if the z-score (or standard-score) exceeds a certain threshold. Here, z-score = (x - mean) / standard\_deviation

Typically, we look for outliers beyond 2 or 3 standard deviations.

## Template

[Association for Computing Machinery (ACM) - SIG Proceedings Template - Overleaf, Online LaTeX Editor](https://www.overleaf.com/latex/templates/association-for-computing-machinery-acm-sig-proceedings-template/bmvfhcdnxfty)

Report Overleaf: <https://www.overleaf.com/5461575522jgqypdfkbxnw>

## To Do

1. Data Pre-processing:
   1. Time split in data (Remove month and year columns)
   2. Time split in data at user level (Keep users with >10 reviews)
2. Models:
   1. Always predict mean (Baseline)
      1. Per state:
         1. MSE: 1.674599470569062
         2. MAE: 1.0676916003156327
      2. Per category:
         1. MSE: 1.6730224281399606
         2. MAE: 1.0670154491063117
      3. Per time period:
         1. MSE: 1.6616142544726031
         2. MAE: 1.0711356377752497
      4. Temporal
         1. Per state:
            1. MSE: 2.529536257440224
            2. MAE: 1.311941193521718
         2. Per category:
            1. MSE: 2.5179072453465023
            2. MAE: 1.3120127629591345
   2. Always predict popular (Baseline) based on number of reviews of a particular business
      1. Per state:
         1. MSE: 2.3346104545019317
         2. MAE: 1.180809783216159
      2. Per category:
         1. MSE: 1.801757653179724
         2. MAE: 1.0675538201975983
      3. Per time period:
         1. MSE: 2.6878508862592203
         2. MAE: 1.198020007462217
      4. Temporal
         1. Per state:
            1. MSE: 2.8617612596606503
            2. MAE: 1.3497097511635285
         2. Per category:
            1. MSE: 2.7617843749970294
            2. MAE: 1.3474911553252638
   3. Linear Regression:
      1. MSE: 0.4172057593484093
      2. MAE: 0.35666849362469816
      3. Temporal:
         1. MSE : 0.3609521795576084
         2. MAE: 0.27726044928267546
   4. Ridge Regression (Fix regularization alpha in Ridge regression):
      1. MSE: 0.4171984394646411
      2. MAE: 0.3565116228860154
      3. Temporal:
         1. MSE : 0.3609566892698407
         2. MAE : 0.277227866375908
   5. Random Forest Regression:
      1. MSE: 0.4245
      2. MAE: 0.3616
      3. Temporal:
         1. MSE: 0.36589232
         2. MAE: 0.27512476
   6. XGBoost Regression [DataTechNotes: Regression Example with XGBRegressor in Python](https://www.datatechnotes.com/2019/06/regression-example-with-xgbregressor-in.html):
      1. MSE: 0.4173
      2. MAE: 0.3555
      3. Temporal:
         1. MSE: 0.36252784
         2. MAE: 0.27383018
   7. BPR With LF
      1. 1.673983892405506 (mse)
      2. 1.0695566110287056 (mae)
      3. Temporal:
         1. MAE : 1.3139664711321033
         2. MSE : 2.535442192195376

* 1. BPR:
     1. Test ---- 1.5905193631324783 (mse)
     2. 1.023491638415881 , (mae)
     3. Temporal:
        1. MAE : 1.3101507897618776
        2. MSE : 2.554251615817853
  2. Tensorflow DNN:
     1. MSE: 0.4166
     2. MAE: 0.36540286
     3. Temporal:
        1. MSE:0.36148914
        2. MAE: **0.27371245**

1. Evaluation Metrics:
   1. MAE
   2. RMSE / MSE
   3. R-square / Adjusted R-square

## References

1. [Degeneracy (graph theory) - Wikipedia](https://en.wikipedia.org/wiki/Degeneracy_(graph_theory))
2. [Language Detection — polyglot 16.07.04 documentation](https://polyglot.readthedocs.io/en/latest/Detection.html#mixed-text)
3. <https://piazza.com/class/l86no3fbyqd2j3/post/458>
4. [Basic regression: Predict fuel efficiency | TensorFlow Core](https://www.tensorflow.org/tutorials/keras/regression)

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense (Dense) (484576, 512) 43520

dropout (Dropout) (484576, 512) 0

dense\_1 (Dense) (484576, 256) 131328

dropout\_1 (Dropout) (484576, 256) 0

dense\_2 (Dense) (484576, 64) 16448

dropout\_2 (Dropout) (484576, 64) 0

dense\_3 (Dense) (484576, 16) 1040

dropout\_3 (Dropout) (484576, 16) 0

dense\_4 (Dense) (484576, 1) 17

=================================================================

Total params: 192,353

Trainable params: 192,353

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Manas Work : 2 models + literature review

Shyam Work : Conclusion + SS something

We need to show our model weights in Linear regression, Ridge regression in report

Sandy : All models test MAE

## Cold start by returning mean

Temporal - RNN

**Mean prediction everytime :-**

**MSE -> 2.537388663186102**

**MAE -> 1.3142678735510411**

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra,

Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan

Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah.

2016. Wide&Deep Learning for Recommender Systems. arXiv:1606.07792 [cs.LG]

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks

for YouTube Recommendations. In Proceedings of the 10th ACM Conference on

Recommender Systems (Boston, MA, USA) (RecSys’16). ACM, New York, NY, USA,

191–198. https://doi.org/10.1145/2959100.2959190

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng

Chua. 2017. Neural Collaborative Filtering. arXiv:1708.05031 [cs.IR]

[15] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for

Implicit Feedback Datasets. In 2008 Eighth IEEE International Conference on Data

Mining (Pisa, Italy). IEEE Computer Society, USA, 263–272. https://doi.org/10.

1109/ICDM.2008.22

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization

Techniques for Recommender Systems. Computer 42, 8 (Aug. 2009), 30–37.

https://doi.org/10.1109/MC.2009.263

[21] Daniel D Lee and H Sebastian Seung. 1999. Learning the parts of objects by

non-negative matrix factorization. Nature 401, 6755 (1999), 788–791. https:

//doi.org/10.1038/44565

[22] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations:

Item-to-Item Collaborative Filtering. IEEE Internet Computing 7, 1 (Jan.

2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344

[34] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee

Kumthekar, Zhe Zhao, LiWei, and Ed Chi. 2019. Sampling-Bias-Corrected Neural

Modeling for Large Corpus Item Recommendations. In Proceedings of the 13th

ACM Conference on Recommender Systems (Copenhagen, Denmark) (RecSys’19).

ACM, New York, NY, USA, 269–277. https://doi.org/10.1145/3298689.3346996

[35] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-

Scale Parallel Collaborative Filtering for the Netflix Prize. In Algorithmic Aspects

in Information and Management, Rudolf Fleischer and Jinhui Xu (Eds.). Springer

Berlin Heidelberg, Berlin, Heidelberg, 337–348. https://doi.org/10.1007/978-3-

540-68880-8\_32