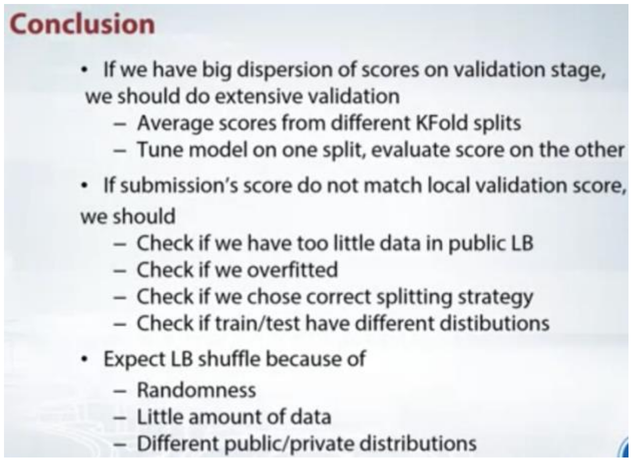
Data splitting strategies

- If in training dataset contains fish photos from three boats, and the testing dataset contains fish photos from the fourth boat, then should use boat-based grouped cross validation. If there is no boat ID column, then first cluster and create that column.
- Train/val split should mimic train/test split.
- In a product search relevance competition, the test set contained completely new search terms. So we couldn't use either a random split or a search term-based split for validation. First split favored more complicated models, which led to overfitting while second split, conversely, to underfitting. So in order to select optimal models, it was crucial to mimic the ratio of new search terms from train/test split.

Problems occurring during validation

Calculate standard deviation of k-fold validation and see if the leaderboard score is expected. If it is not, there can be two reasons:

- Leaderboard data size is too low (Solution: Just trust local k-fold)
- Leaderboard dataset has a different distribution. Solution:
    A. Build a validation set that is most similar to the testing set.
    B. If the leaderboard score is MSE, we submit two submissions, the first one full of a constant number, the second one full of another constant number, and therefore we can calculate the mean of the leaderboard dataset. Since we have the training dataset mean, we can shift the testing prediction up or down by the train-test mean difference. This is called "Leaderboard probing"



Conclusion

- If we have big dispersion of scores on validation stage, we should do extensive validation
  - Average scores from different KFold splits
  - Tune model on one split, evaluate score on the other
- If submission's score do not match local validation score, we should
  - Check if we have too little data in public LB
  - Check if we overfitted
  - Check if we chose correct splitting strategy
  - Check if train/test have different distibutions
- Expect LB shuffle because of
  - Randomness
  - Little amount of data
  - Different public/private distributions

Basic data leaks

- Meta data
- Information in IDs
- Row number

Leaderboard probing and examples of rare data leaks

- There are clusters of data, each cluster has the same label, and each cluster is divided into train/public/private. Therefore, probing the public will reveal the label of the whole cluster.
- For a binary classification task being evaluated by log loss metric, we can probe to find the positive / negative label proportion of the private, assuming private and public has the same distribution. Then we can build the validation data using the proportion.

Metrics Optimization Motivation

Some metrics cannot be optimized efficiently. That is there is no simple enough way to find, say, the optimal hyperplane. That is why sometimes we need to train our model to optimize something different than competition metric. But in this case we will need to apply various heuristics to improve competition metric score.

Regression Metrics Review

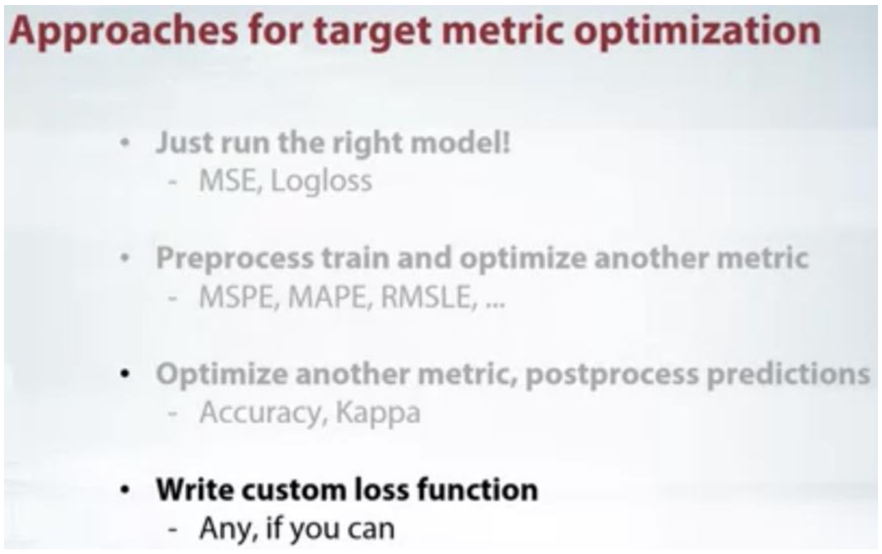
- When predict all value as a constant, the R-squared should be 0.
- In work, if there are outliers, we should use MAE as the evaluation metric; if there are unexpected values that we still care about, we should use MSE.

Classification Metrics Review

- Accuracy score is hard to optimize. Accuracy score gives hard predictions, unlike log loss (i.e. cross-entropy loss).
- Predict every thing as the majority class will result in 0.5 AUC

Metrics Optimization

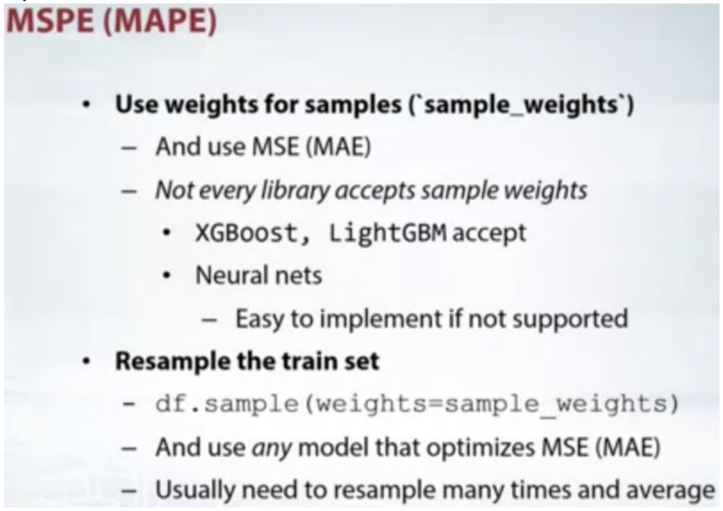
- The loss/objective/cost function is a function that our model optimizes, and the target metric is how we want the solution to be evaluated. For example, log loss is widely used as an optimization loss, while the accuracy score is how the solution is eventually evaluated.
- Sometimes we are lucky and the model can optimize our target metric directly. For example, for MSE metric. So the loss function is the same as the target metric. And sometimes we want to optimize metrics that are really hard or even impossible to optimize directly. In this case, we usually set the model to optimize a loss that is different to a target metric, but after a model is trained, we use hacks and heuristics to negate the discrepancy and adjust the model to better fit the target metric.
- Four approaches and their explanations:



Approaches for target metric optimization

- **Just run the right model!**
  - MSE, Logloss
- **Preprocess train and optimize another metric**
  - MSPE, MAPE, RMSLE, ...
- **Optimize another metric, postprocess predictions**
  - Accuracy, Kappa
- **Write custom loss function**
  - Any, if you can

- MSE and Logloss can be optimized directly.
- For metrics that cannot be optimized directly, we can pre-process the train set and use a model with a loss function which is easy to optimize. For example, while MSPE (which is a weighted version of MSE) cannot be optimized directly with XGBoost, we can resample the train set and optimize MSE loss instead, which XGBoost can optimize.

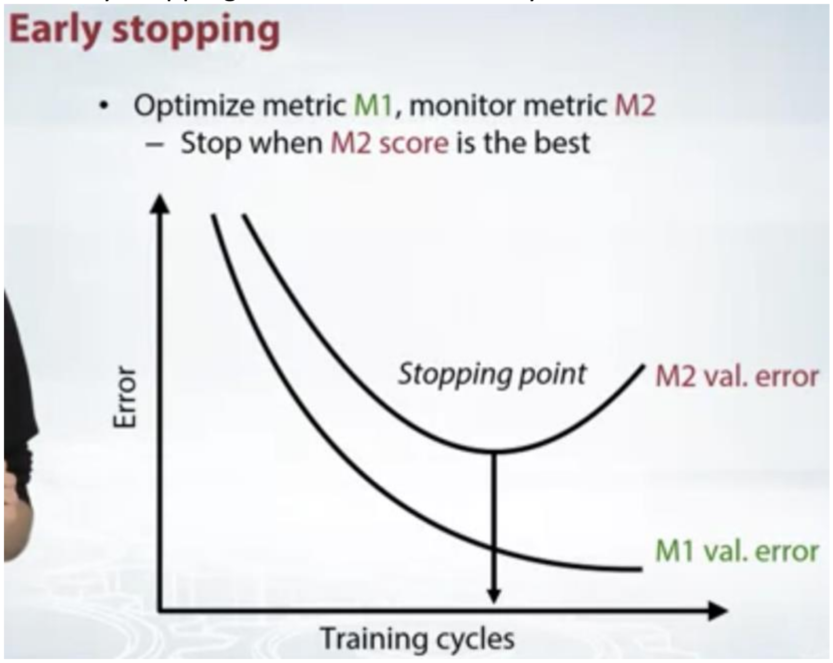


MSPE (MAPE)

- **Use weights for samples (`sample_weights`)**
  - And use MSE (MAE)
  - *Not every library accepts sample weights*
    - `XGBoost, LightGBM` accept
    - Neural nets
      - Easy to implement if not supported
- **Resample the train set**
  - `df.sample(weights=sample_weights)`
  - And use *any* model that optimizes MSE (MAE)
  - Usually need to resample many times and average

- Sometimes, we can optimize incorrect metric, but then post-process the predictions to fit classification, to fit the target metric better.
  - You cannot directly optimize Accuracy. Solution:
    - If it is a binary classification task, fit any metric, and tune with the binarization threshold.
    - If it is a multi-class task, fit any metric and tune parameters comparing the models by their accuracy score, not by the metric that the models were really optimizing. So this is kind of early stopping and the cross validation, where you look at the accuracy score.

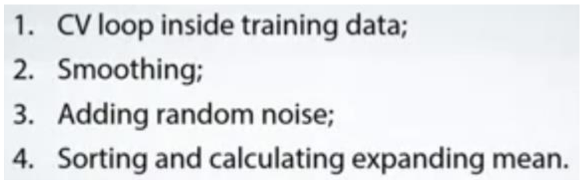
        - AUC is best optimized by pairwise loss, because it is the probability of a pair of the objects to be ordered in the right way. But in practice, most people still use logloss as an optimization loss without any more post processing.
    - Sometimes it's possible to implement a custom loss function which will serve as a nice proxy for the target metric. For example, it can be done for quadratic-weighted Kappa. To define a custom loss function for XGBoost, we need to implement a single function that takes predictions and the target values, and compute first and second-order derivatives of the loss function with respect to the model's predictions. The loss function should be smooth enough and have well-behaved derivatives.
- Early Stopping is a method that always works


Early stopping

- Optimize metric M1, monitor metric M2
  - Stop when M2 score is the best

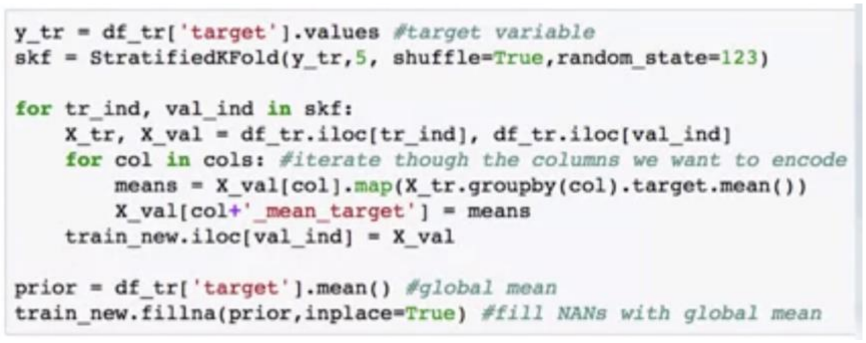
Mean Encoding

For a given data point, we don't want to use target variable of that data point, but estimate the encoding using the rest of dataset.


1. CV loop inside training data;
2. Smoothing;
3. Adding random noise;
4. Sorting and calculating expanding mean.

- In below, CV Loop. Notice that

each CV partition is encoded by the rest of CV partitions.
If there are rare categories, they are encoded by global mean.

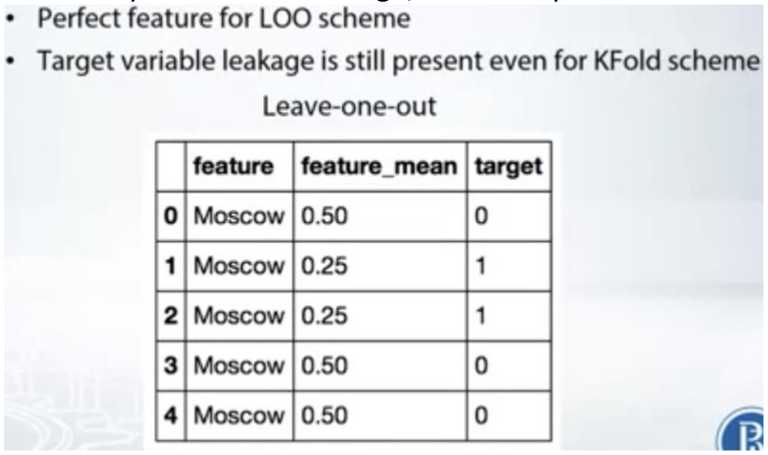


```
y_tr = df_tr['target'].values #target variable
skf = StratifiedKFold(y_tr,5, shuffle=True,random_state=123)

for tr_ind, val_ind in skf:
    X_tr, X_val = df_tr.iloc[tr_ind], df_tr.iloc[val_ind]
    for col in cols: #iterate though the columns we want to encode
        means = X_val[col].map(X_tr.groupby(col).target.mean())
        X_val[col+'_mean_target'] = means
    train_new.iloc[val_ind] = X_val

prior = df_tr['target'].mean() #global mean
train_new.fillna(prior,inplace=True) #fill NANs with global mean
```

There may still be data leakage, but usually it is fine.



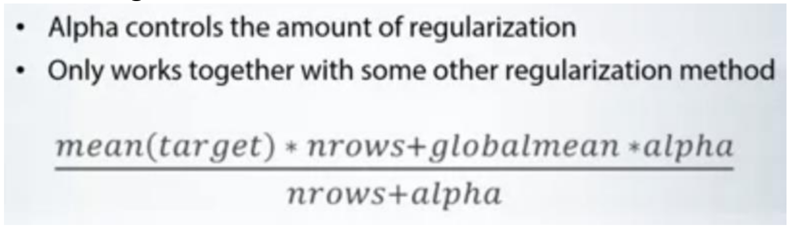
- Perfect feature for LOO scheme
- Target variable leakage is still present even for KFold scheme

Leave-one-out

	feature	feature_mean	target
0	Moscow	0.50	0
1	Moscow	0.25	1
2	Moscow	0.25	1
3	Moscow	0.50	0
4	Moscow	0.50	0

- In below, smoothing

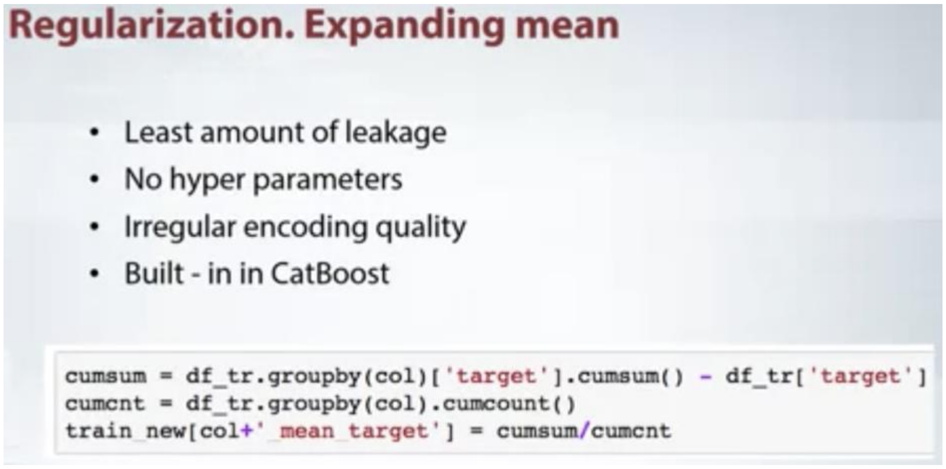
Smoothing won't work on its own. Need to combine it with CV loop regularization.



- Alpha controls the amount of regularization
- Only works together with some other regularization method

$$\frac{mean(target) * nrows + globalmean * alpha}{nrows + alpha}$$

- Adding random noise is not easy to work well.
- In below, the CatBoost

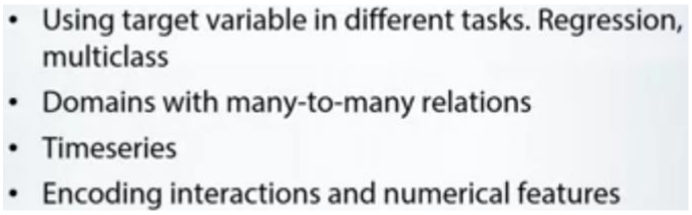


Regularization. Expanding mean

- Least amount of leakage
- No hyper parameters
- Irregular encoding quality
- Built - in in CatBoost

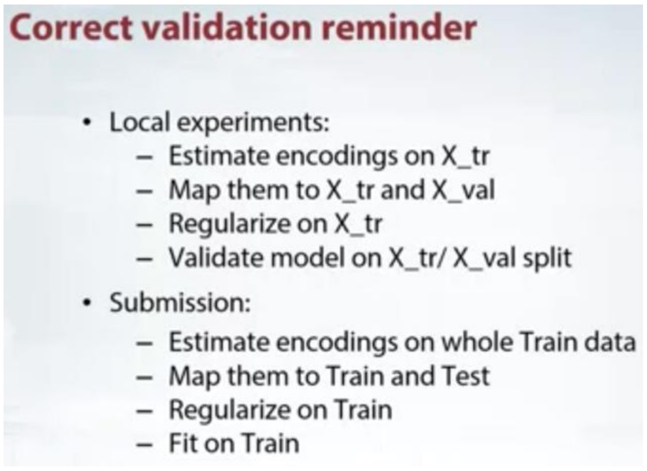
```
cumsum = df_tr.groupby(col)['target'].cumsum() - df_tr['target']
cumcnt = df_tr.groupby(col).cumcount()
train_new[col+'_mean_target'] = cumsum/cumcnt
```

More on Mean Encoding



- Using target variable in different tasks. Regression, multiclass
- Domains with many-to-many relations
- Timeseries
- Encoding interactions and numerical features

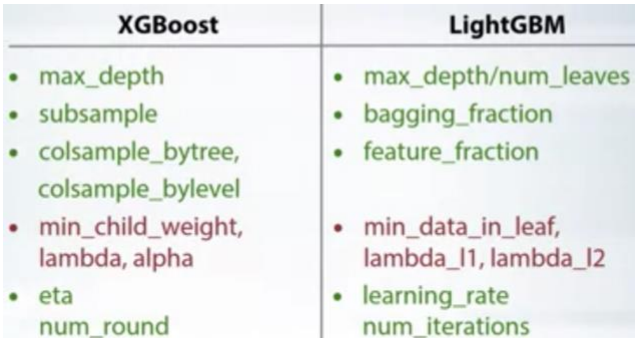
- In regression tasks, we can try a variety of statistics, like medium, percentile, standard deviation of target variable. Unlike binary classification where a mean is the only meaningful statistic we can extract from target variable. We can even calculate some distribution bins. For example, if target variable is distributed between 1 and 100, we can create 10 bin features. In the first feature, we'll count how many data points have targeted between 1 and 10, in the second, between 10 and 20, and so on.
- For multi-class tasks, meaning encoding is: for every feature we want to encode, we will have n different encodings where n is the number of classes. There is additional advantage, as tree models usually solve multi-class task in one versus all fashion, and now the mean encoding for multi-class introduces some additional information about the structure of other classes.
- Don't even think about estimating encodings before splitting the data.



Correct validation reminder

- Local experiments:
  - Estimate encodings on X_tr
  - Map them to X_tr and X_val
  - Regularize on X_tr
  - Validate model on X_tr/ X_val split
- Submission:
  - Estimate encodings on whole Train data
  - Map them to Train and Test
  - Regularize on Train
  - Fit on Train

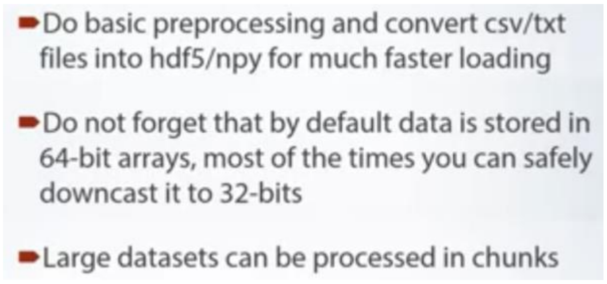
Hyperparameter Tuning



XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction
• colsample_bytree, colsample_bylevel	• feature_fraction
• min_child_weight, lambda, alpha	• min_data_in_leaf, lambda_l1, lambda_l2
• eta num_round	• learning_rate num_iterations

- For NN,
  - larger batch size, more likely to overfit
  - batch size to increase alpha time allows learning rate to also increase alpha times, just as a rule of thumb

Data Loading



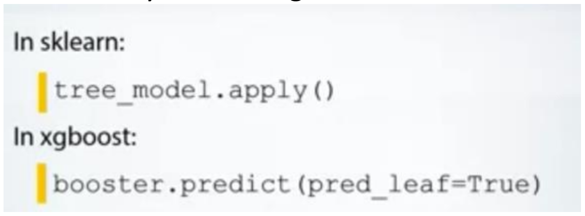
- Do basic preprocessing and convert csv/txt files into hdf5/npy for much faster loading
- Do not forget that by default data is stored in 64-bit arrays, most of the times you can safely downcast it to 32-bits
- Large datasets can be processed in chunks

- HDF5 for Panda's dataframes
- MPI for non bit arrays

- Running experiment often require a lot of kernel restarts, which leads to reloading all the data. And loading class csv files may take minutes while loading data from HDF5 or MPI formats is performed in a matter of seconds.
- By default, Panda store data in 64-bit arrays, which is unnecessary in most of the situations. Downcasting everything to 32 bits will result in two-fold memory saving.
- Panda's support out of the box data relink by chunks, via chunks size parameter in read_csv function. So most of the data sets may be processed without a lot of memory.

Some Feature Engineering Ideas

- Neighbourhood data (think about house price)
- Get Min and Max for a group of rows contributed by one customer
- Matrix factorizations to reduce too many features created (it mentioned NMF works well for tree based model and for count data)
- How to use directly extract high order interactions from a tree based model:

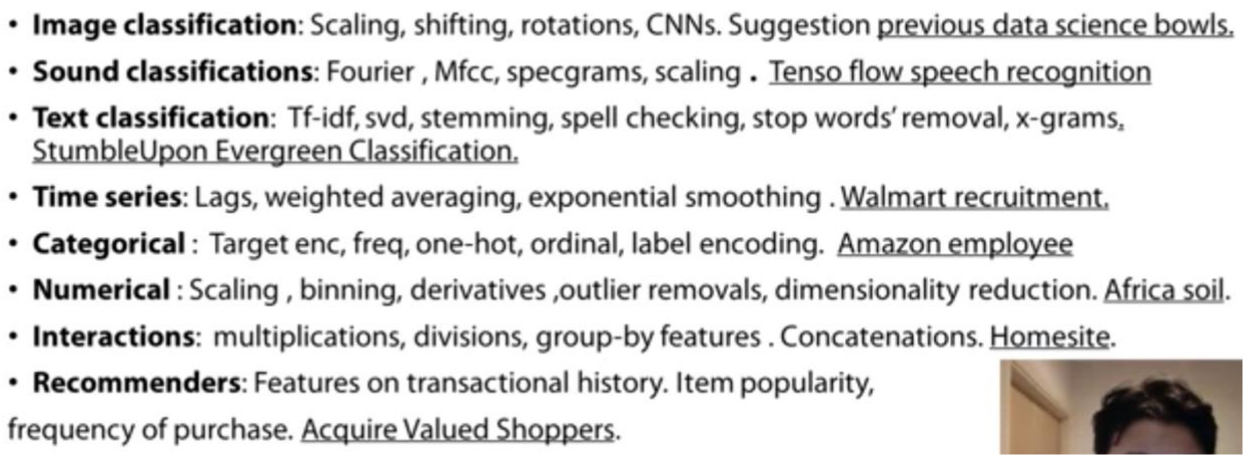


In sklearn:

```
tree_model.apply()
```

In xgboost:

```
booster.predict(pred_leaf=True)
```



- **Image classification**: Scaling, shifting, rotations, CNNs. Suggestion previous data science bowls.
- **Sound classifications**: Fourier , Mfcc, specgrams, scaling **.** Tenso flow speech recognition
- **Text classification**: Tf-idf, svd, stemming, spell checking, stop words' removal, x-grams, StumbleUpon Evergreen Classification.
- **Time series**: Lags, weighted averaging, exponential smoothing . Walmart recruitment.
- **Categorical** : Target enc, freq, one-hot, ordinal, label encoding. Amazon employee
- **Numerical** : Scaling , binning, derivatives ,outlier removals, dimensionality reduction. Africa soil.
- **Interactions**: multiplications, divisions, group-by features . Concatenations. Homesite.
- **Recommenders**: Features on transactional history. Item popularity, frequency of purchase. Acquire Valued Shoppers.