How to Win a Data Science Competition: Lean From Top Kagglers

Coursera - National Research University Higher School of Economics

2020

Chapter 1

week1

1.1 Introduction and Recap

- In Kaggle competitions, looks for **Kernels** and/or **Notebooks** - people share code that way and you can learn from them.
- IMPORTANT: Insight is more important than algorithm
- IMPORTANT: Don't limit yourself - use advanced feature enginnering; run huge greedy calculations over night.
- Recap of models/methods:
  - Linear Methods
    - Logistic Regression; Support Vector Machine
    - GOOD FOR: Sparse, high dimensional data
    - BAD: Often linear separability is not a good approximation
  - Tree Based Methods
    - Decision Tree; Random Forrest; Gradient Boosted Decision Trees (GBDT)
    - GOOD FOR: good default method for tabular data; good for non-linear relationships
    - BAD FOR: hard to capture linear separation (say diagonal line in a 2D plane)

      from sklearn: can create over-complex trees that do not generalise data well - that is they are prone to overfitting

      trees can't extrapolate trends - they are good at interpolation not extrapolation
    - *Sklearn* has good random forrest; *xgboost* has good GBDT
  - K-Nearest neighboors (KNN)

    good for feature building
  - Neural Networks (NN)

    good for images, sounds, texts and sequences

    very good framework is *pytorch*
- Most powerfull methods currently are GBDS and NN
- No free lunch theorem - there is no single methods that beats all other methods for all applications
- TODO: Look into H20 - open source, in-memory, distributed, fast and scalable machine learning and predictive analytics platform.

  Looks quite interesting!
- Hardware: SSD is critical; use cloud computing for anything too big.

1.2 Feature Preprocessing

- VIP: Strong connection between preprocessing and model choice
- Scaling: $x_i \to \alpha x_i$
  - decision trees are not affected by a scaling tx, but non-tree based methods like NN, kNN and regularized linear models are very affected

    in KNN - $x \to 0 * x$ means ignore x, while $x \to \infty x$ means x dominates

    Any methods that relies on gradient descent is sensitive to scaling (logistic regression, SVM, neural networks, etc. etc)

  - appraoch 1: Normalization (map to [0,1])

$$x \rightarrow (x - x.min())/(x.max() - x.min)$$

    code: sklearn.preprocessing.MinMaxSaler
  - approach 2: standardization (to $\mu = 0, \sigma = 1$)

$$x \rightarrow (x - x.mean())/x.std()$$

    code: sklearn.preprocessing.StandardScaler
  - Can use scaling to boost or attenuate signals - so scaling can be thought as just another hyperparameter to tune
    In general, start with all features in the same scale and explore changing that.

- Outliers
  - linear models extremely sensitive
  - approach 1: **Winsorization** - clip values between percentiles
    code: UPPER, LOWER = np.percentile(x, [1,99]); y = np.clip(x, UPPER, LOWER)
    very common in finance
  - approach 3: Rank
    good for knn and linear models when you don't have time to handle outliers by hand
  - approach 3: Other transformation
    log transformation; raising to power ¡ 1 - these have the benefit of bringing outliers in closer, and spreading things around zero apart

- Missing values
  - can be Nans, empty strings, outliers like -1, or -999
  - identifiying when a value actually represents a missing value can be challenging. Main tool: Histogram.
  - Sometimes missing values contain alot of useful info - there might be a reason the value is missing
    So it is good to create new features like 'isMissing' or something like that.
  - 3 main imputation techniques
    * NaN $\rightarrow$ value outside range
      Gives trees possibilities to use this; Performance for linear models suffers greatly
      Curated data often comes with somethign like this already (as described above)
    * NaN $\rightarrow$ mean, median or mode.
      good for linear methods; not good for trees
    * Try to reconstruct NaN
      not easy; need to know something about the data generating process.
  - WARNING: be vary careful with early NaN imputation if you then build features based on imputed feature
    for exampl if you fill a cyclic feature with median values, and then construct diff as new feature - will lead to prominent discontinuities and flat zones.
  - some libraries, like XGBoost can handle NaNs automatically

- For all all of these, you must learn the transformation from training data, and then apply the same one on testing. Sklearn Transformation API (used by the transformations above) allows you to do this. VERY GOOD.

- USEFUL TRICK: create different predictors from same feature using different preprocessing techniques.

- USEFUL TRICK: Mix models that are trained on different preprocessed versions of the data

1.3 External reading

- FROM SKLEARN
  - when data is sparse, you don't want a scaler that moves zeros. Think about nature of your data to chose scalar.
  - Transfromer API: scaler = sklearn.Preprocessing.StandardScaler().fit(X_train); x_test_scaled = scaler(X_test)
  - many estimators in sklearn expect features to look more or less $\sim$ N(0,1)
  - VIP: Note l1 and l2 regularizes assume all features centered around zero and have variance in the same order.

- other useful tx:
    - preprocessing.Normalizer() - scaling individual samples to have unit norm (useful when estimating pair similarity via dot prod)
    - preprocessing.OneHotEncoder(), preprocessing.OrdinalEncoder()
    - preprocessing.KBinDiscretizer(n_bins, encode).fit() - different strategies available

      histograms focus on counting features in particular bin, whereas dsicretizers focus on labeling features with bin

- Sebastian Raschka
  - Tree based classification is probably the only scale invariant algo
  - VIP: when using PCA, it is better to standardize (mean 1, std 0) than just normalize (map to [0,1]) - scaling affects covariance

    cool example doing PCA and then bayes classifier on top - compar prediction score.

- Jason Brownlee - Discover feature engineering: "Most algorithms are standard - we spend most of our efforts on feature engineering"
- Rahul Agarwal - Best practices in feature engineering
  - LogLoss clipping technique: clip prediction prob to [0.05. 0.95] when using log loss metric - it penalizes heavily being very certain and wrong
  - Use PCA for rotation, not only dim rec
  - sometimes add interaction features, A*B, A/B, A+B, A - B

1.4 Feature Generation

- encodings categorical/ordinal
  - label encoding: map categories to 1,2,3,4...

    good for tree based methods, not good for knn or linear (because it assume order and proportionality in labels - moreover dependence is likely not-linear)

    code: sklearn.preprocessing.LabelEncoder() or pd.factorize()
  - Frequency encoding: map to numerical value representing frquency of category

    can help trees use less splits

    this is even sometimes useful in linear models
  - one-hot encoding: create indicator variable for each category

    these can be good for linear methods, but in general slow down methods (explosion of features) and might not help (specially trees)

    though will help tree if target depends on lbael encoded feature in a very non linear way

    code: pd.get_dummies() or sklearn.preprocessing.OneHotEncoder()
- Datetime and coordinates
  - Very different than simple numerical or categorical because we can interpret their meaning - they have much context
  - dates and times can lead to two main types of features
    - moments in a period (ie using periodicity of datetime)

      ex: day of week, day of month, month in year, etc. Or minute value, hour value, etc.

      useful to capture repetitive pattern in data
    - time since/to event

      can be row independent: years since 2000, for all rows (so all rows have same reference)

      row dependent: days since last holiday (or till next holiday) - two rows can be referring to different holidays
  - very useful to diffs between two date columns
  - once you generate new features, numerical or categorical, preprocess them accordingly
  - coordinates
    - typically you want to calculate distance to points of interest (nearest hospital, school, etc)
    - very useful to calculate aggregated statistics for objects around an area

      ex: # of flats around a point -¿ proxy for popularity of area

      ex: mean price of flats around a point -¿ gives sense of price of area.

- Collection of tricks
  - separte price into integer part and fractional part - can utilize differences in peoples *perception* of a price
  - Create feature, 'isRoundNumber' - people often use numers like 5 and 10, while robots can use many decimals.
  - One-hot encode interaction between categorical features [just concatenate strings and OHE result]

    Not so useful in tree based models, because they can easily approximate this with individual categories.
  - Sometimes simple multiplication, or division of features makes a huge difference.

    linear models can't approximate these, and trees have a very hard time approximating them.
  - sometimes useful to add slightly rotated coordinate system - particularly when using trees.

    ex: if a particular street happens to be a good division, but that street is not algined with coordinates, then tree uses many split to approximate.

    hard to know what rotation to use a priori - so add several and check effect.

1.5 Feature Extraction From Texts and Images

- For Text...
- Preprocessing: 1) lowercase, 2) lemmatization, 3) stemming, 4) remove stopwords
- feature extraction: Bag of words - column per word in corpus, row perd doc, count ocurrences

  can extend to n-grams, of either words of letters
- Post processing: TFiDF (Term Frequency and Inverse document frequency)
- OR ...
- embeddings (word2vec, or others)
  - Still use preprocessing
  - create vector representation of words in text
  - uses nearby words (unsupervised)
  - often resulting vector space has interpretable operations
  - training can take a long time - check for pretrained
- BOW and Word2Vec often give very different results - use bott together
- for images
  - look for pretained models and do some fine tunning
  - use image augmentation to increase training samples (crops, rotations, adding noise, etc.)

    reduces overfitting

1.6 Questions

- in GBM_drop_tree notebook - why are raw predictions - the output of the staged_decision_function - approaching ± 8? (I assume it has to do with depth 3 choice of trees, but still, shouldn't output be close to ± 1 which is actual y-values?