Week 1: Recap

Ameya Godbole

1 Introduction

A thanks to:

1. the course instructors: Dmitry Ulyanov, Alexander Guschin, Mikhail Trofimov, Dmitry Altukhov and Marios Michailidis
2. National Research University Higher School of Economics
3. Coursera

The broad goal of the course is to get the necessary knowledge and expertise to succesfully participate in data science competitions.

2 Competition Mechanics

1. **Data:** Consists of provided data and (optionally) data from external sources
2. **Model:** Anything that produces answers from data. Must be the **best possible predictor** and **reproducible**
3. **Submission:** The prediction (optionally code)
4. **Evaluation:** Computed score using prediction and correct answer
5. **Leaderboard:** Relative position among participants

During the competition, score is evaluated only on a *'public'* subset of the test data. Final standings are evaluated on the remaining *'private'* test set.

3 Recap of basic ML algorithms

3.1 Model Families

1. Linear
   - E.g. logistic regression, SVM
   - Good for sparse high dimensional data
2. Tree-based
   - E.g. decision tree, random forest, GBDT (gradient boosted decision trees)
   - Complicated models use decision tree as building block
   - Decision tree uses divide-and-conquer approach to recursively split the space. Data is split in boxes having samples of same type
3. kNN-based
   - Intuition: Closer objects are likely to have the same label
   - Heavily relies on measure of point closeness

4. Neural networks

3.2 No free lunch theorem

"There is no method which outperforms all others for all tasks"

or

"For every method, we can construct a task for which this particular method will not be the best"

The reason is that every method relies on some assumptions about data or task. If the assumptions fail, the method will perform poorly.

3.3 Suggested reading

- Overview of methods[1]
  1. Scikit-Learn (or sklearn) library
  2. Overview of k-NN (sklearn's documentation)
  3. Overview of Linear Models (sklearn's documentation)
  4. Overview of Decision Trees (sklearn's documentation)
  5. Overview of algorithms and parameters in H2O documentation
- Additional Tools[1]
  1. Vowpal Wabbit repository
  2. XGBoost repository
  3. LightGBM repository
  4. Frameworks for Neural Nets: Keras,PyTorch,TensorFlow,MXNet, Lasagne
  5. Example from sklearn with different decision surfaces
  6. Arbitrary order factorization machines

4 Software/Hardware requirements

StandCloud Computing:

- AWS, Google Cloud, Microsoft Azure

AWS spot option:

- Overview of Spot mechanism
- Spot Setup Guide

Stack and packages:

- Basic SciPy stack (ipython, numpy, pandas, matplotlib)
- Jupyter Notebook
- Stand-alone python tSNE package
- Libraries to work with sparse CTR-like data: LibFM, LibFFM

[1] List obtained from course page

- Another tree-based method: RGF (implemetation, paper)
- Python distribution with all-included packages: Anaconda
- Blog "datas-frame" (contains posts about effective Pandas usage)

5 Feature preprocessing and generation

Choice of preprocessing and generation pipeline depends on:

Feature types: Different types require diff preprocssing techniques and the feature generation method also vary accordingly with type.

Model: *No free luch theorem* can be extended to understand that features generated may benefit one model but be useless for another.

Oftentimes it is beneficial to train model on concatenated dataframe generated by different preprocessings or to mix models trained on differently preprocessed data.

Feature generation can involve:

1. use of prior knowledge and logic
2. digging into data; creating and checking hypotheses

5.1 Numeric Features

Scale:

- Scale of features does not impact tree-based models but has a large impact on non tree-based models. In the extremes, the model may ignore/give undue importance to a feature only due to its scale. E.g.
  - for kNN, a feaure showing very small scale(range) will be ignored while features with large scale will be given importance
  - scale of features affects regularization and stability of gradient descent in neural networks
- Preprocessing tools:
  1. To [0,1] (sklearn.preprocessing.MinMaxScaler)

     $\hat{X} = \frac{X - X.min()}{X.max() - X.min()}$
  2. To mean=0,std=1 (sklearn.preprocessing.StandardScaler)

     $\hat{X} = \frac{X - X.mean()}{X.std()}$

  With these scaling, impact of all features on non-tree-based models will become roughly similar. Feature impact can then be tuned to try and improve performance for models like kNN.

Outliers:

- Outliers in terms feature value can affect decision boundary/function, especially for linear models.
- At the same time, abnormal target value for few samples can also skew the model.
- Preprocessing tools:
  1. **Clipping** values to an upper and lower bound set as some percentile of the given data. This is a well known method for financial applications (*Winsorization*). e.g. setting lower bound to value at 5th percentile and upper bound to value at 95th percentile (90% winsorization)

2. **Rank transformation** (scipy.stats.rankdata) sets the distances between proper sorted values to be equal. It is better than minmaxscaler at handling outliers as spacing between all values is set equal. E.g.

   $rank([-100, 0, 1e5]) \rightarrow [0, 1, 2]$

   $rank([1000, 1, 10]) \rightarrow [2, 0, 1]$

   If there is no time for manual outlier handling, linear models, kNN and neural networks may benefit from this transformation. To apply rank transformation to test data, transformation must be stored. Alternatively, apply rank transformation to concatenated train and test data.

3. **Log transform** ($np.log(1+x)$) and raising to **power ¡ 1** (e.g. $np.sqrt(x+2/3)$) can help non tree-based models especially neural networks. They drive outlier values closer to the average and improve resolution near 0.

Feature generation case studies: Intuitive features help not only linear models and neural networks but also GBDT which has trouble approximating multiplication and division.

1. Price and area → price per sq. area
2. horizontal and vertical distance → direct distance
3. Some features help to make distinctions between robots and humans explicit. E.g. in applications like auction price estimation where humans generally tend to set rounded bids and not numbers like, say, 1457. As another example, humans do not reply at exact time intervals which may be a useful feature in spambot detection.

5.2 Categorical and Ordinal Features

Label encoding For tree-based methods, most of the information in categorical variables can be extracted by mapping them to numbers. Does not work for non tree-based models. There are following type of encoding:

1. alphabetical (sklearn.preprocessing.LabelEncoder)
2. order-of-appearance (pandas.factorize)
3. frequency-based i.e. use their frequency in data as encoding. Even non-tree-based values can use this feature if frequency and target value are correlated.

One-hot encoding For non-tree-based this is the suggested method to encode categorical data (pandas.get_dummies, sklearn.preprocessing.OneHotEncoder). This method introduces one new column per category. This is not recommended for tree-based models because it introduces too many new features and can obstruct trees from using the numerical features if outnumbered. Since only one entry will be 1 per row(sample), to efficiently store the data, we need sparse matrices. Going with sparse matrices makes sense if number of non-zero values is far less than half the total number of values.

Ordinal features can be encoded by 'cumulative' encoder[2]. E.g.

poor	1	0	0	0
fair	1	1	0	0
good	1	1	1	0
excellent	1	1	1	1

The first column can then be dropped as it is always 1. Quote: "*This encoding is very useful, especially when you think about how regularization affects the coefficients associated with each category in linear models. So, the bias term will provide the prior "score" for the observation. Then, each coefficient provides the gap between classes, being the regularization associated with a small difference between contiguous classes instead of small coefficients in the original space.*"

Feature generation One of the basic ways to generate new categorical features is to create a new feature representing the interactions of other categorical features. For example, if gender [male,female]

[2]Suggested by Kelwin Fernandes in the course forum

and class[1,2,3] are available, generate new feature as [1male,2male,3male,1female,2female,3female]. This explicit feature improves learning.

5.3 Date and time Features

Periodicity Useful to capture repetitive patterns in the data. E.g. number of weeks, days, hours, minutes within the time the data were collected or until the next event

Time since Two types:

1. row-independent: Time from a general moment/date, like the year 2000. All samples become comparable on same time scale
2. row-dependent: E.g. time since last campaign/holiday or time to the next special event

Date difference Subtract 2 event datetimes to generate feature. E.g. difference between last_purchase_date and last_call_date for customer churn prediction task

5.4 Co-ordinate Features

If extra infrastructural data is available, distance to nearest shop, library, hospital, etc. can be added as a feature.

If such data is not available, train and test data can be used to define interest points. For example, by dividing the map into grids and locating the costliest house in each grid, distance to this house can be added as a feature. Alternatively, cluster the available data and use cluster centres as the interst points. A complete cluster may also be defined as important and then distance to the cluster can be added.

Another approach is to calculate aggregate statistics for the neighbouring area as a feature.

Note: Decision trees may benefit from the use of rotated co-ordinates for defining grids.

5.5 Quick Links

- Preprocessing in Sklearn
- Discover Feature Engineering, How to Engineer Features and How to Get Good at It

6 Handling missing values

It is very important to note that missing values may be hidden from us i.e. replaced by something other than NaN.

Imputation methods

1. Replace with a value outside feature range: Useful because it gives trees to put such samples into a separate category. However, performace of neural network or linear model can suffer (basically introduces outlier for them)
2. Replace with mean / median: Reversed effect on models. Better for non-tree-based and harder for tree-based models
3. Try to reconstruct value

A new feature can be immediately added to demarcate rows where a particular feature value was missing. Note that imputed values must be carefully considered for subsequent feature generation as it can have unintended consequences.

Some libraries like XGBoost can handle missing values out-of-the-box. Such approaches can significantly improve performance.

Sometimes it may be beneficial to treat outlier as missing value.

7 Feature Extraction from Text

1. Bag-of-words (sklearn.feature_extraction.text.CountVectorizer)

   Simply count the number of occurences of each word. Post processing may be needed/used to make samples comparable and boost important features. Some practical counting methods are:

   - **Term Frequency:** Word count by total count of the sentence.

     $tf = \frac{1}{x.sum(axis=1)}$

     $x = x * tf$

   - **Inverse Document Frequency:** We scale a word to incorporate the number of documents it appears in.

     $idf = np.log(\frac{x.shape[0]}{(x>0).sum(axis=0)}$

     $x = x * idf$

   - A combination of TF and iDF (sklearn.feature_extraction.text.TfidfVectorizer)
   - **Ngrams:** Count occurences of all sequences of N words to make use of context (sklearn.feature_extraction.text.C with proper arguments)

   Careful preprocessing helps BoW drastically.

   - lowercase
   - stemming: chops of ending of words heuristically and thus unites related words
   - lemmatization: carefully uses vocabulary and knowledge of morphology to unite related words
   - stopwords: remove articles, prepositions and extremely common words

2. Embeddings

   Sparse representations of words (such as word2vec, Glove, FastText, etc) and even documents (doc2vec) have been shown to encode text meaningfully. They demonstrate high-level relations such as gender efficiently. All preprocessing steps used for BoW also apply to the training of embeddings.

BoW	w2v
large vectors	relatively smaller representations
each position interpretable	not interpretable directly but some relations can be shown
	words with similar meaning have similar embedding

7.1 Quick Links

- Feature extraction from text with Sklearn
- Text Classification With Word2Vec
- NLTK
- TextBlob

8 Feature Extraction from Images

CNN outputs of different layers can be used as good image representations. Deeper layers generally provide specialized descriptors for a particular task while earlier layers give task independent low-level features. If data is sufficiently available, the CNN may be trained from scratch for the given task. In case data is insufficient, fine-tuning of a pre-trained model is preferred.