How to win a Kaggle competition

Part 3: Advanced Feature Engineering

Chris Chen@Calgary Data Science Academy

Mean encoding categorical features

Mean encoding

- Encode high-cardinality categorical features (e.g. postal code) by learning the probability of the dependent variable with Empirical Bayes.
- Most effective encoding approach for high-cardinality categorical features
- Model agnostic. Works for both classification and regression problems.
- Prone to overfitting thus requires extra caution when use it.
- Using out-of-fold scheme to avoid overfitting. This was not mentioned in the original paper but turned out to be a MUST!

Suppose we are working on a binary classification with a training dataset where *x_p* is postal code and *y* is the dependent variable.

- Prior prob of y: $P(y) = sum(y)/len(y)$
- Posterior prob of y: $P(y|x\_p) = mean(y).groupby(x\_p)$
- Encoded x_p: $P\_enc(y|x\_p) = \lambda * P(y) + (1- \lambda) * P(y|x\_p)$
  - $\lambda = 1/ (1 + \exp((n-k)/f)$, where
  - n is the number of occurrences of the corresponding categorical value
  - k and f are hyper parameters that control the smoothness of the encoding.

Mean Encoding: A Preprocessing Scheme for High-Cardinality Categorical Features, Daniel M.
http://helios.mm.di.uoa.gr/~rouvas/ssi/sigkdd/sigkdd.vol3.1/barreca.pdf

Word embedding: representing words by their context

- A word's meaning is given by the words that frequently appear close-by
  - "You shall know a word by the company it keeps" (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

Industrial applications of	***deep learning***	to large-scale speech recognition started around 2010
can be thought of as a representational layer in a	***deep learning***	architecture that transforms an atomic word into a positional representation
These issues may possibly be addressed by	***deep learning***	architectures that internally form states homologous to image-grammar

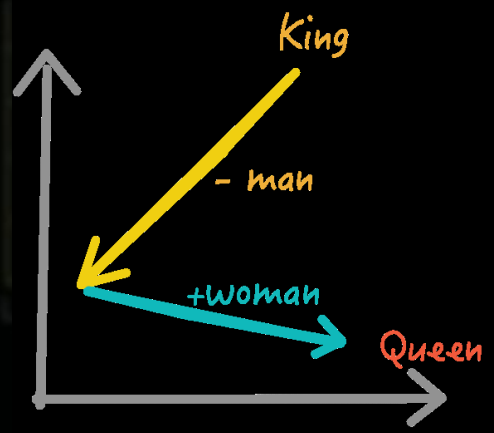
The word "*deep learning*" can be represented by the context words around it.

Retrieved from http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf

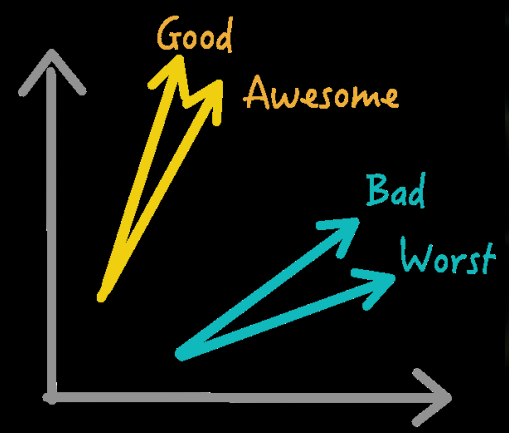
Word2vec overview

word2vec (Mikolov et al. 2013) is a framework for learning word vectors:

- Idea:
  - We have a large corpus of text
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word *c* and context ("outside") words *o*
  - Use the similarity of the word vectors for c and o to calculate the probability of *o* given *c* (or vice versa)
  - Keep adjusting the word vectors to maximize this probability.



Analogy



Similarity

- word2vec encodes each word into a *n* (typically 300) dimension vector.
- The vector representations allow us to perform mathematical operations between words.
- One can use pretrained word2vec embedding or train a word2vec embedding model using custom corpus.

From word2vec to item2vec

Items can be embedded using the same approach for word embedding, which is particularly useful when working with transaction data (user events, credit card purchases etc.)

Step 1: generate "sentences"

```
manager_by_building = full_data.groupby('building_id')['manager_id'].apply(list)
manager_by_building.head()
```

```
building_id
0                                [98e13ad4b495b9613cef886d79a6291f, 23a01ea7717...
00005cb939f9986300d987652c933e15                 [f19288238987b18a693e16ee23720c20]
00024d77a43f0606f926e2312513845c    [8d425b08cdf4dec134bf5b714ebac267, b37f269315c...
000ae4b7db298401cdae2b0ba1ea8146    [7a3fa7c61b497b5713aff16b18e1bfa8, 7a3fa7c61b4...
0012f1955391bca600ec301035b97b65                 [f398f307789fc70a80e152a96d02712a]
Name: manager_id, dtype: object
```

Step 2: train a embedding model

```
from gensim.models import FastText
manager_model = FastText(size=10, window=3, min_count=1, workers=16)
manager_model.build_vocab(sentences=manager_by_building)
manager_model.train(sentences=manager_by_building.values,
                    total_examples=len(manager_by_building.values), epochs=5)
```

Step 3: encode items with the trained embedding model

```
manager_emb = full_data['manager_id'].apply(lambda x:manager_model[x]).values
manager_emb = np.array([e.reshape(1,-1) for e in manager_emb]).reshape(-1,10)
manager_emb
```

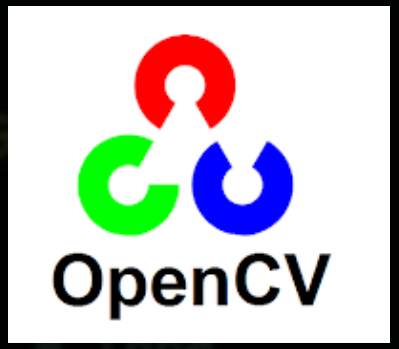
```
C:\Users\cb7vm\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_la
ted `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() i
  """Entry point for launching an IPython kernel.
```

```
array([[ 0.84920025, -1.7292897 ,  0.72283334, ..., -0.15048704,
        -1.1680835 ,  0.17687915],
       [-1.6523657 , -2.8142295 ,  1.127939  , ...,  1.054796  ,
         1.1928889 , -0.7064095 ],
       [-0.9340583 , -0.33655065,  0.86684597, ...,  0.5717648 ,
         0.9733493 ,  0.8709165 ],
       ...,
       [-1.4670557 , -0.9655638 ,  0.37873507, ...,  0.6895176 ,
         0.8991395 ,  0.13892737],
       [-1.3679405 , -1.5596694 ,  0.41887194, ...,  0.7045937 ,
         0.6333788 ,  0.1113212 ],
       [-0.64208645, -1.2125679 ,  0.24885689, ...,  0.41439584,
         0.15499695, -0.00752767]], dtype=float32)
```

Extracting features from images

Raw image

OpenCV



Classic image attributes: size, dullness, whiteness, colors, blurness, uniformity

Pre-trained CNN model

- Image Class: television room: 0.280 living room: 0.163
- Image attributes: enclosed area, no horizon, man-made, glass, wood, reading, natural light, soothing, glossy
- Apply one-hot encoding on predicted image class
- Apply tf-idf on image attributes

Object detection (yolo, ssd)

- Outputs: chair: 40%, potted plant: 25%, vase: 75%, tv monitor: 99%
- Create a list of objects: [chair, potted-plant, vas, tv-monitor]
- Create sparse features by applying tf-idf on all the list of objects

Think out of the box

Data forensic

When working on the competition many participants including me found a webpage on renthop's website in regard to an interesting system called "HopScore":
HopScore: a proprietary algorithm that sorts and ranks all apartment listings on our site based on a set of different factors. These factors fall into three categories:

- Listing quality
  - Photos
  - Descriptions
- Manager Performance
  - Reply rate
  - Activeness
- Listing freshness
  - A listing's freshness will decrease over time
  - A listing may occur many times in the dataset and the interest level may be different

Relevance	Listing Quality	Manager Performance	Listing Freshness
bathrooms	Fair	Fair	Low
bedrooms	Fair	Fair	Low
building_id	Fair	Fair	Low
created	Low	Fair	High
description	High	High	Low
display_address	High	Fair	Low
features	High	High	Low
latitude	High	Fair	Fair
listing_id		Fair	High
longitude	High	Fair	Fair
manager_id	Fair	High	Low
photos	High	Fair	Low
price	High	Fair	Fair
street_address	High	Fair	Low

Key findings:

- Manager seems the most important factor
- One apartment could be listed for multiple times with different listing ids.
- Question: how can we develop a pseudo "apartment_id to uniquely identify apartment?
- Can we use the concatenation of Location + bedrooms + bathrooms + description ?

How to hack hopscore?

- Listing quality
  - % of upper case words
  - # of '*', '!', '$', '<' etc
  - HTML tags
  - Phone number in desc
  - email address in desc
  - is street address the same as display address
- Listing freshness
  - How many times a property had been listed?
  - How long since a property had been listed?
- Manager Performance
  - count/mean/median/std/min/max of bathrooms/bedrooms/lon/lat/price/number of photos/number of features/ length of desc group by manager id (key to success)
  - Number of buildings managed by the manager
  - Range of listings
  - Manager activeness - number of active days
  - Manage responsiveness - mean of durations between listings
- Pseudo apartment id:
  - Hash the concatenation of building id, bedrooms, bathrooms, description and features.

Location! Location! Location!

- Perform k-means to cluster listings into geo-segments
  - Aggregate attributes by geo-segments
  - Calculate distance between a listing and the centroid of a cluster
- Create clusters after filtering certain keywords such as “supermarket”, “subway”, “bus”, “park” etc
  - Aggregate attributes by geo-segments
  - Calculate distance between a listing and the centroid of a cluster
- Not allowed by the competition but you may want to give it a try from knowledge learning perspective:
- Retrieve postal code/ neighbourhood by lat/lon, perform similar aggregations mentioned before
- Combine public demographic stats by joining on postal code

The magic feature

Magic feature, or data leakage is very common topic in both Kaggle competitions and real-world projects. Nevertheless, "magic feature" hunting is part of the game at Kaggle and oftentimes it's a key to winning a competition. It turned out a magic feature was discovered by a Grand Master who, however, chose to keep it with him. Later, it was found by another Grand Master KazAnova who decided to share it with the community.

- The magic feature: timestamp of listing photos
- One would have won a gold medal by simply adding the magic feature to a bronze medal solution.
- Why it works so magically?
  - It provides additional temporal information
  - But more importantly, it is likely the "fingerprint" of an apartment!

Feature engineering progress

Feature engineering	logloss	Would have ranked in this competition
Basic feature engineering	0.5324	Top 25%
+ target encoding	0.5296	Top 24%
+ manager performance	0.5287	Top 23%
+ location	0.5251	Top 21%
+ item2vec	0.5250	Top 20%
+ magic feature	05124	Top 10%

* Cross validation was done with LightGBM.
** Rank was estimated based on the assumed CV/LB gap.