Back To Module Home

Machine Learning System Design

0% completed

Machine Learning Primer

- Introduction
- Feature Selection and Feature Engineering
- Training Pipeline
- Inference
- Metrics Evaluation

Video Recommendation

Feed Ranking

Ad Click Prediction

Rental Search Ranking

Estimate Food Delivery Time

Machine Learning Knowledge

Machine Learning Model Diagnosis

Conclusion

Mark Module as Completed

Feature Selection and Feature Engineering

Learn how tech companies like Facebook, Twitter, and Airbnb design their feature selection and feature engineering to serve billions of users.

We'll cover the following

- 1. One hot encoding
  - Common problems
  - Best practices
  - One hot encoding in tech companies
- 2. Feature hashing
  - Benefits
    - Feature hashing example

- Feature hashing in tech companies
- 3. Crossed feature
  - Crossed feature in tech companies
- 4. Embedding
  - Benefits
  - How to generate/learn embedding vector?
  - Embedding in tech companies
- 5. Numeric features
  - Normalization
  - Standardization

1. One hot encoding#

One hot encoding is a very common technique in feature engineering. It converts categorical variables into a one-hot numeric array.

- One hot encoding is very popular when you have to deal with categorical features that have medium cardinality.

One hot encoding example

Common problems#

- Expansive computation and high memory consumption are major problems with one hot encoding. High numbers of values will create high-dimensional feature

vectors. For example, if there are one million unique values in a column, it will produce feature vectors that have a dimensionality of one million.

- One hot encoding is not suitable for Natural Language Processing tasks. Microsoft Word's dictionary is usually large, and we can't use one hot encoding to represent each word as the vector is too big to store in memory.

Best practices#

- Depending on the application, some levels/categories that are not important, can be grouped together in the "Other" class.
- Make sure that the pipeline can handle unseen data in the test set.

> In Python, there are many ways to do one hot encoding, for example, `pandas.get_dummies` and sklearn `OneHotEncoder`. `pandas.get_dummies` does not "remember" the encoding during training, and if testing data has new values, it can lead to inconsistent mapping. `OneHotEncoder` is a Scikitlearn Transformer; therefore, you can use it consistently during training and predicting.

One hot encoding in tech companies#

- It's not practical to use one hot encoding to handle large cardinality features, i.e., features that have hundreds or thousands of unique values. Companies like Instacart and DoorDash use more advanced techniques to handle large cardinality features.

2. Feature hashing#

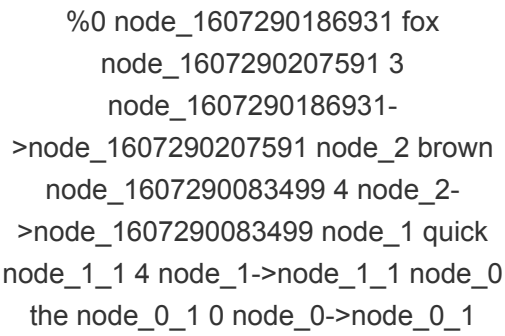
Feature hashing, called the hashing trick, converts text data or categorical attributes with high cardinalities into a feature vector of arbitrary dimensionality.

Benefits#

- Feature hashing is very useful for features that have high cardinality with hundreds and thousands of unique values. Hashing trick is a way to reduce the increase in dimension and memory by allowing multiple values to be present/encoded as the same value.

Feature hashing example#

- First, you chose the dimensionality of your feature vectors. Then, using a hash function, you convert all values of your categorical attribute (or all tokens in your collection of documents) into a number. Then you convert this number into an index of your feature vector. The process is illustrated in the diagram below.



An illustration of the hashing trick for desired dimensionality of 5 for the originality of K of values of an attributes

- Let's illustrate what it would look like to convert the text "The quick brown fox" into a feature vector. The values for each word in the phrase are:

```
the = 5
quick = 4
brown = 4
fox = 3
```

- Let define a hash function, h, that takes a string as input and outputs a non-negative integer. Let the desired dimensionality be 5. By applying the hash function to each word and applying the modulo of 5 to obtain the index of the word, we get:

```
h(the) mod 5 = 0
h(quick) mod 5 = 4
h(brown) mod 5 = 4
h(fox) mod 5 = 3
```

- In this example:
  - `h(the) mod 5 = 0` means that we have one word in dimension **0** of the feature vector.
  - `h(quick) mod 5 = 4` and `h(brown) mod 5 = 4` means that we have two words in dimension **4** of the feature vector.
  - `h(fox) mod 5 = 3` means that we have one word in dimension **3** of the feature vector.
  - As you can see, that there are no words in dimensions 1 or 2 of the vector, so we keep them as 0.
- Finally, we have the feature vector as: `[1, 0, 0, 1, 2]`.
- As you can see, there is a collision between words “quick” and “brown.” They are both represented by dimension 4. The lower the desired dimensionality, the higher the chances of collision. To reduce the probability of collision, we can increase the desired dimensions. This is the trade-off between speed and quality of learning.

> Commonly used hash functions are MurmurHash3, Jenkins, CityHash, and MD5.

Feature hashing in tech companies#

- Feature hashing is popular in many tech companies like Booking, Facebook, Yahoo, Yandex, Avazu and Criteo.
- One problem with hashing is collision. If the hash size is too small, more collisions

will happen and negatively affect model performance. On the other hand, the larger the hash size, the more it will consume memory.

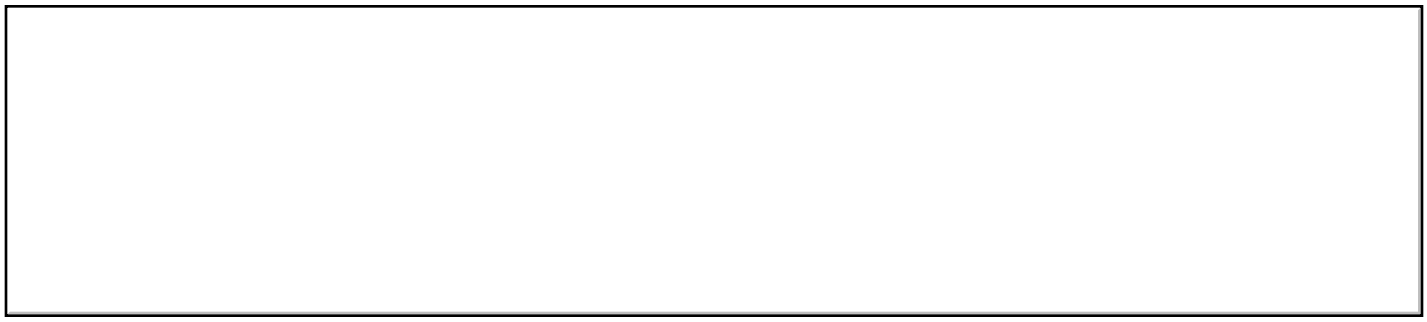
- Collisions also affect model performance. With high collisions, a model won't be able to differentiate coefficients between feature values. For example, the coefficient for "User login/ User logout" might end up the same, which makes no sense.

> Depending on application, you can choose the number of bits for feature hashing that provide the correct balance between model accuracy and computing cost during model training. For example, by increasing hash size we can improve performance, but the training time will increase as well as memory consumption.

3. Crossed feature#

- Crossed features, or **conjunction**, between two categorical variables of cardinality `c1` and `c2` is just another categorical variable of cardinality `c1` × `c2`. If `c1` and `c2` are large, the conjunction feature has high cardinality, and the use of the hashing trick is even more critical in this case. Crossed feature is usually used with a hashing trick to reduce high dimensions.

- As an example, suppose we have Uber pick-up data with latitude and longitude stored in the database, and we want to predict demand at a certain location. If we just use the feature latitude for learning, the model might learn that a city block at a particular latitude is more likely to have a higher demand than others. This is similar for the feature longitude. However, a feature cross of longitude by latitude would represent a well-defined city block. Consequently, the model will learn more accurately.



Uber Pickups Map in 2015 (source Uber)

> Read more about different techniques in handling latitude/longitude here: Haversine distance, Manhattan distance.

Crossed feature in tech companies#

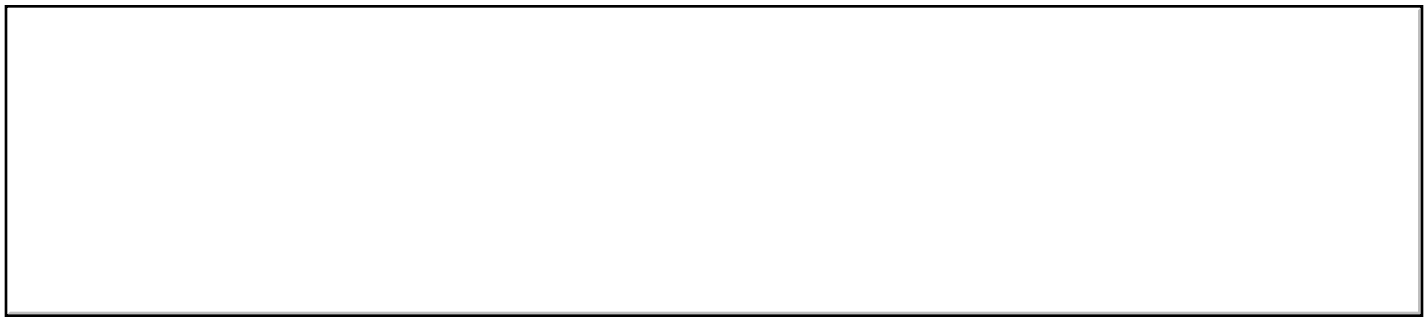
This technique is commonly applied at many tech companies. LinkedIn uses crossed features between user location and user job title for their Job recommendation model. Airbnb also uses cross features for their Search Ranking model.

4. Embedding#

Feature embedding is an emerging technique that aims to transform features from the original space into a new space to support effective machine learning. The purpose of embedding is to capture semantic meaning of features; for example, similar features will be close to each other in the embedding vector space.

Benefits#

- Both one hot encoding and feature hashing can represent features in another dimension. However, these representations do not usually preserve the semantic meaning of each feature. For example, in the **Word2Vector** representation, embedding words into dense multidimensional representation helps to improve the prediction of the next words significantly.



A 4-dimensional embedding

- As an example, each word can be represented as a `d` dimension vector, and we can train our supervised model. We then use the output of one of the fully connected layers of the neural network model as embeddings on the input object. For example, `cat` embedding can be represented as a `[1.2, -0.1, 4.3, 3.2]` vector.

How to generate/learn embedding vector?#

- For popular deep learning frameworks like TensorFlow, you need to define the dimension of embedding and network architecture. Once defined, the network can learn embedding automatically. For example:

```
# Embed a 1,000 word vocabulary into 5
dimensions.
embedding_layer =
tf.keras.layers.Embedding(1000, 5)
```

Embedding in tech companies#

This technique is commonly applied at many tech companies.

- Twitter uses Embedding for UserIDs and cases like recommendations, nearest neighbor searches, and transfer learning.

- Doordash uses Store Embedding (store2vec) to personalize the store feed. Similar to word2vec, each store is one word and each sentence is one user session. Then, to generate vectors for a consumer, we sum the vectors for each store they ordered from in the past 6 months or a total of 100 orders. Then, the distance between a

store and a consumer is determined by taking the cosine distance between the store's vector and the consumer's vector.

- Similarly, Instagram uses account embedding to recommend relevant content (photos, videos, and Stories) to users.

> The embedding dimensionality is usually determined experimentally or from experience. In TensorFlow documentation, they recommend: $d = \sqrt[4]{D}$. Where D is the number of categories. Another way is to treat d as a hyperparameter, and we can tune on a downstream task.

> In large scale production, embedding features are usually pre-computed and stored in key/value storage to reduce inference latency.

5. Numeric features#

Normalization#

- For numeric features, normalization can be done to make the mean equal 0, and the values be in the range [-1, 1]. There are some cases where we want to normalize data between the range [0, 1].

  $$v = \frac{v - min\_of\_v}{max\_of\_v - min\_of\_v}$$

  where,

  v is feature value,

  min_of_v is a minimum of feature value,

  max_of_v is a maximum of feature value

Standardization#

- If features distribution resembles a normal distribution, then we can apply a standardized transformation.

  $v = \frac{v - mean\_of\_v}{std\_of\_v}$

  where,

  v is feature value,

  $mean\_of\_v$ is a mean of feature value,

  std_of_v is the standard deviation of feature value

- If feature distribution resembles power laws, then we can transform it by using the formula:

  $log(\frac{1+v}{1+median\_of\_v})$

> In practice, normalization can cause an issue as the values of `min` and `max` are usually outliers. One possible solution is "clipping", where we choose a "reasonable" value for `min` and `max`. You can also learn more about how companies apply feature engineering here.

Feature selection and Feature engineering quiz
Q We have a table with columns UserID, CountryID, CityID, zip code, and age. Which of the following feature engineering is suitable to represent the data in a Machine Learning

algorithm?

Introduction

Next

Training Pipeline

Mark as Completed

Report an Issue