Feature Engineering - Handling Cyclical Features

SOHCAHTOA amirite?

Posted by David Kaleko (http://davidkaleko.com/#about) on Mon 30 October 2017

I was browsing twitter yesterday (follow me! (https://twitter.com/davidkaleko)) when I came across this tweet (https://twitter.com/BecomingDataSci/status/924814225637367808) by Data Science Renee (https://twitter.com/BecomingDataSci) linking to this Medium article (https://medium.com/towards-data-science/top-6-errors-novice-machine-learning-engineers-make-e82273d394db) called "Top 6 Errors Novice Machine Learning Engineers Make" by Christopher Dossman (https://medium.com/@cdossman). This drew my attention because I'm somewhat new to the field (and even if I weren't, it's always worth reviewing the fundamentals (http://blog.davidkaleko.com/fundamentals-regression-classification.html)). While I'm not guilty of all six novice mistakes, I'm certainly not innocent either. The most interesting mistake to me was:

"Not properly dealing with cyclical features"

Christopher writes:

> *Hours of the day, days of the week, months in a year, and wind direction are all examples of features that are cyclical. Many new machine learning engineers don't think to convert these features into a representation that can preserve information such as hour 23 and hour 0 being close to each other and not far.*

Yup. I've tried to predict my fair share of quantities using cyclical features incorrectly. Christopher points out that proper handling of such features involves representing the cyclical features as (x,y) coordinates on a circle. In this blog post, I'll explore this feature engineering task and see if it really improves the predictive capability of a simple model.

To begin, let's download a public dataset that has some cyclical qualities. I found a bi*cycle* sharing dataset online (pardon the double entendre) which includes some basic features, with the aim of predicting how many bikes are being used at any given hour. Let's download, unzip, and have a quick look.

In [1]:

```
!curl -O 'https://archive.ics.uci.edu/ml/machine-learning-databases/00275/Bike-Sharir
!mkdir 'data/bike_sharing/'
!unzip 'Bike-Sharing-Dataset.zip' -d 'data/bike_sharing'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Ti
                                 Dload  Upload   Total   Spent    Le
100  273k  100  273k    0     0   185k      0  0:00:01  0:00:01 --:-
Archive:  Bike-Sharing-Dataset.zip
  inflating: data/bike_sharing/Readme.txt
  inflating: data/bike_sharing/day.csv
  inflating: data/bike_sharing/hour.csv
```

In [2]:

```
import pandas as pd

df = pd.read_csv('data/bike_sharing/hour.csv')
print df.columns.values
```

```
['instant' 'dteday' 'season' 'yr' 'mnth' 'hr' 'holiday' 'weekday'
 'workingday' 'weathersit' 'temp' 'atemp' 'hum' 'windspeed' 'casual'
 'registered' 'cnt']
```

It looks like there are a bunch of features in here that are likely valuable to predict

cnt, the count of users riding bikes (probably the sum of "casual" riders and "registered" riders). Let's have a look at the *mnth* (month) feature, and the *hr* (hour) feature and try to transform them *a la* Mr. Dossman.

In [3]:

```
df = df[['mnth','hr','cnt']].copy()
```

In [4]:

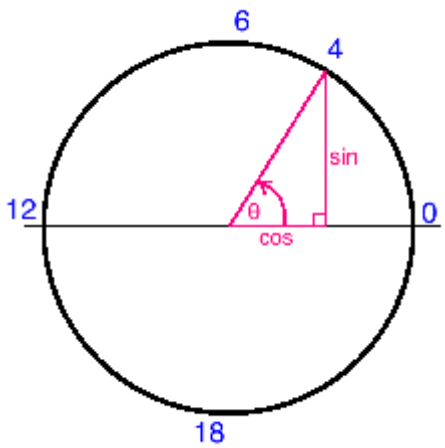
```
print 'Unique values of month:',df.mnth.unique()
print 'Unique values of hour:',df.hr.unique()
```

```
Unique values of month: [ 1  2  3  4  5  6  7  8  9 10 11 12]
Unique values of hour: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
```

So far, so logical. Months are numbered one through twelve, and hours are numbered 0 through 23.

The Magic

Now the magic happens. We map each cyclical variable onto a circle such that the lowest value for that variable appears right next to the largest value. We compute the x- and y- component of that point using *sin* and *cos* trigonometric functions. You remember your unit circle, right? Here's what it looks like for the "hours" variable. Zero (midnight) is on the right, and the hours increase counterclockwise around the circle. In this way, 23:59 is very close to 00:00, as it should be.



Note that when we perform this transformation for the "month" variable, we also shift the values down by one such that it extends from 0 to 11, for convenience.

In [6]:

```
import numpy as np

df['hr_sin'] = np.sin(df.hr*(2.*np.pi/24))
df['hr_cos'] = np.cos(df.hr*(2.*np.pi/24))
df['mnth_sin'] = np.sin((df.mnth-1)*(2.*np.pi/12))
df['mnth_cos'] = np.cos((df.mnth-1)*(2.*np.pi/12))
```

Now instead of hours extending from 0 to 23, we have two new features "hr_sin" and "hr_cos" which each extend from 0 to 1 and combine to have the nice cyclical characteristics we're after.

The claim is that using this transform will improve the predictive performance of our models. Let's give it a shot!

Impact on Model Performance

To begin, let's try to use just the nominal hours and month features to predict the number of bikes being ridden. I'll use a basic sklearn neural network (http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html) and see how well it performs with K-fold cross validation. The loss function I'll use is (negative) mean squared error (https://en.wikipedia.org/wiki/Mean_squared_error). I'll also use a standard scaler in an sklearn Pipeline, though it probably isn't necessary given the range in values of our two features.

In [8]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import Pipeline

# Construct the pipeline with a standard scaler and a small neural network
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('nn', MLPRegressor(hidden_layer_sizes=(5,), max_iter=1000)))
model = Pipeline(estimators)

# To begin, let's use only these two features to predict 'cnt' (bicycle count)
features = ['mnth','hr']
X = df[features].values
y = df.cnt

# We'll use 5-fold cross validation. That is, a random 80% of the data will be used
# to train the model, and the prediction score will be computed on the remaining 20%.
# This process is repeated five times such that the training sets in each "fold"
# are mutually orthogonal.
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kfold = KFold(n_splits=5)

results = cross_val_score(model, X, y, cv=kfold, scoring='neg_mean_squared_error')
print 'CV Scoring Result: mean=',np.mean(results),'std=',np.std(results)
```

```
CV Scoring Result: mean= -31417.4341504 std= 12593.3781285
```

That's a pretty large average MSE, but that's the point. The neural network struggles to interpret month and hour features because they aren't represented in a logical (cyclical) way. These features are more categorical than numerical, which is a problem for the network.

Let's repeat the same exercise, but using the four newly engineered features we created above.

In [9]:

```
features = ['mnth_sin','mnth_cos','hr_sin','hr_cos']
X = df[features].values

results = cross_val_score(model, X, y, cv=kfold, scoring='neg_mean_squared_error')
print 'CV Scoring Result: mean=',np.mean(results),'std=',np.std(results)
```

```
CV Scoring Result: mean= -23702.9269968 std= 9942.45888087
```

Success! The average MSE improved by almost 25% (remember this is *negative* MSE so the closer to zero the better)!

To be clear, I'm not saying that this model will do a good job at predicting the number of bikes being ridden, but taking this feature engineering step for the cyclical features definitely helped. The model had an easier time interpreting the engineered features. What's nice is that this feature engineering method not only improves performance, but it does it in a *logical* way that humans can understand. I'll definitely be employing this technique in my future analyses, and I hope you do too.

Love this post? Hate this post? Think I'm an idiot? Let me know in the comments below! I'm always trying to learn new things and hone my data science skills so any feedback is welcome!

Comments !

18 Comments kalekoblog 1 Login

Recommend 7 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Franco • 4 months ago

Maybe obvious to many, but the sin/cos technique also preserves information in hours (23 and 0 close), days (Sunday and Monday close) ...which can be important for IoT running 24/7.

4 ^ v • Reply • Share ›

mbruner63 • 2 years ago

Great article. I love the sin/cos method of dealing with cyclical features.

3 ^ v • Reply • Share ›

Mariel G • 2 years ago

Nice explanation!
When you define the cyclical features, I would say a "2" is missing, it should be:
df['hr_sin'] = np.sin(df.hr*(2*np.pi/24))
df['hr_cos'] = np.cos(df.hr*(2*np.pi/24))
df['mnth_sin'] = np.sin((df.mnth-1)*(2*np.pi/12))
df['mnth_cos'] = np.cos((df.mnth-1)*(2*np.pi/12))

2 ^ v • Reply • Share ›

David Kaleko Mod ➚ Mariel G • 2 years ago

Hey Mariel! Thanks for the kind words.

I initially thought that too! After trying it out, I saw that including the 2 doesn't map the features how I want.

Let's try your formula on the "hours" variable. For hour == 0, sin(0*(2pi/24)) = 0. Then, for hour == 12, sin(12*(2pi/24)) also = 0. I want midnight on the opposite side of the unit circle as noon. It turns out I want to use half the period of the trig function here, and 2*pi includes the full period.

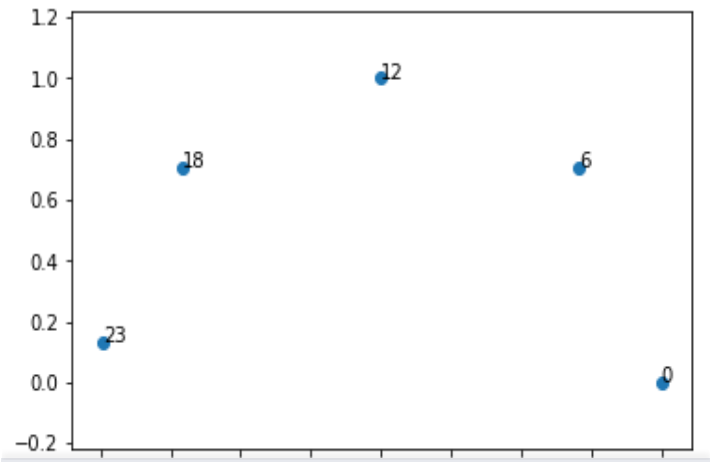
*** EDIT *** It turns out Mariel is correct on this one! I've fixed the formulas in the post above. Thanks, Mariel!

1 ^ v • Reply • Share ›

Mariel G ➚ David Kaleko • 2 years ago

Hi David! I get what you mean, I think it's confusing because of the cyclical nature of the functions. But, the way I see it, in case of the hours variable, if you want 23 to be close to 0, and 12 to be on the opposite side of the unit circle, I would use 2*pi ...

This is what you get with "pi":



see more

^ v • Reply • Share ›

Show more replies

Franco • 4 months ago

hi David, thanks for sharing your knowledge. I need to test it, but this sounds genius!

1 ^ v • Reply • Share ›

Tobias • 2 years ago

Thanks for this post! Concise explanation of both the problem and the solution. I reproduced your results and just for fun, compared them to OneHot-encoded month/hour variables. The result for OneHot is similar to the nominal version, so I'll use your "cyclical" transformation from now on :-)

1 ^ v • Reply • Share ›

David Kaleko Mod ➚ Tobias • 2 years ago

Thanks for reproducing the results, Tobias! I hadn't considered onehot-encoding these variables. I'll point out that in this example "hours" are integers (and therefore onehot-encodable) but this cyclical approach could work for continuous times, too (11:30 am would be 11.5 hours). Thanks for reading!

1 ^ v • Reply • Share ›

Shukman NG • 5 days ago

Thanks for the incredible sharing! May I ask why we do sin and cos in the same time instead of use just either sin or cos?

^ v • Reply • Share ›

Viktor Butorin • 2 months ago

Thank you, sir! I have been struggling with the same dataset and found your suggestion by chance. Brilliant explanation on how to handle cyclical values.

^ v • Reply • Share ›

Florentin Anggraini Purnama • 9 months ago

Great article! By the way, I think it could be better for readers who are not as familiar with trigonometry if you explain why you need to do the normalization with the hours and months values before converting them into angles.

^ v • Reply • Share ›

JuneHao Ching • a year ago

Hi, a noob here.
Would like to ask a simple question ya, when X take in the values, which is
features = ['mnth_sin','mnth_cos','hr_sin','hr_cos']
X = df[features].values
does this mean that X take in 4 values to predict Y(cnt) rather than

the earlier 2 values?

Reply • Share ›

David Kaleko Mod → JuneHao Ching • a year ago

Hi JuneHao --

Yes that is correct. What was previously two features turns into four features after applying this transformation. :)

Reply • Share ›

JuneHao Ching → David Kaleko • a year ago

Thanks David! So normally we will get double the features of original one right?

Reply • Share ›

Show more replies

rojour • 2 years ago

Nice example. Kind of funny that this morning I was reading a post in another forum asking for advice about doing an inventory planning algorithm that would account for cyclical periods...

Reply • Share ›

aquacalc • 2 years ago

 (https://www.linkedin.com/in/davidkaleko)

 (http://twitter.com/davidkaleko)  (http://facebook.com/davidkaleko)

 (http://github.com/kaleko)

Blog powered by Pelican (http://getpelican.com), which takes great advantage of Python (http://python.org).