Audio file

Your Recording 14.wav

Transcript

00:00:10 Speaker 2

The last technique of data transformation that employs some mathematical concept is that of encoding. Now encoding is the process where we have categorical data, in other words, text data, and turning that into numerical data. Because we know that our algorithm works with numbers. So we can't work with words like red or green if we're talking about color of a car, for example, but we could work with numbers like 012. That's going to allow our algorithm to work with them.

00:00:37 Speaker 2

So what types of encoding do we have? We could go with straightforward label encoding. So imagine in my house price data example, I have a feature called neighborhood and I currently have values of downtown, suburbs, and rural as the values that occur in the rows of my data.

00:00:52 Speaker 2

Now what I could do is I could have an equivalent numerical value for each one of those categorical features.

00:00:58 Speaker 2

So an encoded neighborhood feature would be one for downtown, 2 for suburbs, 3 for rural. Ah, OK. Now that I have an encoded label, what I could do is I could drop neighborhood feature altogether because that's a feature that contains categorical data. My algorithm can't work with it, so let's just keep the other one. Let's just keep the new encoded feature only that is purely numeric. That seems pretty good so far.

00:01:19 Speaker 2

So now that we've dropped the original neighborhood categorical feature and we just have our encoded neighborhood, we think we're good and the training process starts in our data. But the algorithm might derive some level of importance in those numbers. It might determine that one is more important than two, or three is more important than two. It might choose to say, well, is one more important than two, or is the numerical increase in value 3 is greater than two, therefore it's more important. In other words, is there an ordinal purpose to those numbers?

00:01:46 Speaker 2

Now we know there's not, but how is a model in training going to be able to determine if that's relevant or not? So we do have a problem here.

00:01:53 Speaker 2

So to overcome that, we often use a technique called 1 hot encoding. Now with one hot encoding, I start out with my neighborhood feature and I synthesize some new features. I'm going to add 3 new features, neighborhood downtown, neighborhood suburbs, and neighborhood rural. And what I'm using these for is kind of like a feature flag to say, well, do I belong to downtown? If I do, I put a one in there. I don't. I put a zero in there. Do I belong to the suburb Neighborhood? No, that'd be a zero. Am I rural? No, that'd be a 0. So it's kind of like a mutual exclusive flag. I set the flag to one for one of those.

00:02:23 Speaker 2

Neighborhoods, that's the equivalent of the categorical data saying downtown. And again, I would drop the neighborhood feature here once I'd encoded this way, but now there is no greater importance of suburbs versus rural versus downtown. It's merely A numeric flag that it belongs to that category and then the algorithm will be able to work out if there's a relationship that contributes to the target.

00:02:44 Speaker 2

So how do I achieve this in code? Well, yet again, we're going to use scikit learn to help us achieve this. Scikit learn is a treasure chest of wonderful utilities that are going to help you. Now notice that as we talk about these techniques, none of them have been specific to Amazon Sagemaker. Sagemaker is just going to be the platform where we run these techniques, but we have to know what these techniques are and when to use them in order to train our model. We can see here in scikit learn we are importing the one hot encoder from the preprocessing library. Then I can see all I need to do is create an instance of the one.

00:03:13 Speaker 2

Coder call the transform method and supply it with input data of my data frame and call out which feature column in my data frame, this case neighborhood, that I want it to perform one encoding upon. Remember this will add additional columns to your data. Once we've got that data encoded, we are then dropping. That's what the second last line there is doing. We are going to drop the original neighborhood column because we don't need that categorical data in there anymore because we've represented it differently with one hot encoded data.

00:03:41 Speaker 2

One, hot encoding is very powerful, but it shouldn't be used everywhere. Let's take a look at an example. Imagine you've got a data set that reflects house prices in a city. And for each data point that you have, you will have post code data. OK, Well, when we think about the number of post codes there are in a city, there could be many 100 if not thousands of post codes. OK, Is that bad? Well, when you have a high range of potential values for a particular feature, then we'd say that that feature has high cardinality. There's high number of potential values.

00:04:11 Speaker 2

OK, well, but why is that bad? Well, we need to think about what 1 encoding actually does.

00:04:16 Speaker 2

Let's look at an example. If I'm one hot encode for post code, then what that would mean is I would get an additional new column in my data set for each and every new post code with a single one digit against the post code that that rule belongs to. But now my data is very wide with many many columns to represent all the potential post codes with just a one flag to say which one that row belongs to. Now that expansion of your data to make it ultra wide is not going to help your algorithm during training. This could lead to long time to do training, long time to converge.

00:04:46 Speaker 2

Ordered to have no convergence at all. It's just it can't workout the pattern of data. There's just too much disparate sparse data.

00:04:52 Speaker 2

So where you have high cardinality of values that are non numeric, we might want to use a different encoding technique.

00:05:00 Speaker 2

What we could do is something called target encoding and we could group the data by the categorical variable. So in the post code we could say, OK, well there's going to 1st part and second part to the post code. We could say, let's group together all the post codes using the prefix. We'll not worry about the suffix. We'll take the prefix, the post code and we'll group, let's say all the properties in that first prefix. What we could then do is say, well, for feature you're trying to capture, you could take the mean of the target variable like house price and you could calculate the mean value for that particular group where the post code.

00:05:30 Speaker 2

That shares the same 1st 4 digits for example.

00:05:32 Speaker 2

OK, And then we would replace the post code altogether with this corresponding mean target value.

00:05:39 Speaker 2

OK, so the post code feature would go away altogether, but we would replace it with a numeric value which would be the mean value for all the house prices within the same approximate area of those sharing the same prefix post code. And this can achieve a very powerful target encoding technique that doesn't massively widen your data set, thus risking convergence.

00:06:00 Speaker 2

So we gain the benefit of encoding, but with a dimensionality reduction we're only going to need one column instead of potentially hundreds more.

00:06:09 Speaker 2

It doesn't do anything to break the relationships between the feature and the target. You've still maintained that linkage.

00:06:15 Speaker 2

And is very efficient because we're not having to add those additional dimensions to represent as a column each and every variant of that single category.

00:06:24 Speaker 2

So how does that look for a data set?

00:06:27 Speaker 2

Well, if we have a data set here of host prices and postcards, and here I've simplified the post code into just two characters like A1 or B2 and I can see OK for a one I've got a House of 300,000 and I've got a house in B2 of 250,000 and oh another house in A1 post code of 320,000. OK, so we could work out what the average house price is for the post code A1 and they do the same for B2, the same for C3 and so forth.

00:06:51 Speaker 2

OK, so now we've worked out. OK, well the average or mean house price for post code A1 is 310,000. The average for post code B2 is 260. The average for C3 post code is 150. OK, so now that we know what those average values are, let's substitute that back in as our new encoded feature, our target encoded feature. So let's do that.

00:07:09 Speaker 2

So now I can see my data, it's called post code. It's got the house price for that data point. And now it's got a target encoded value or feature. So I can see, OK for that first row, the target encoded feature of 310,000, that's the average price for the A1 post code. And you can see a little bit further down, we've got another post code of A1 house price 320,000. But what's the average target encoded for that post code 310 for C3 propose only one data point 150,000. So that's the target encoded one for the C3 post code.

00:07:36 Speaker 2

So by doing this, we preserve the relationship of value to post code, but it means that we'd no longer need the post code itself and therefore we could drop the post code categorical feature and just be left with the numerical feature and we didn't have to add in hundreds more new features.

00:07:51 Speaker 2

So how do we do this in code?

00:07:54 Speaker 2

Well, again, we have our data set. We're using a pandas data frame for this. OK, let's work out a mean of house prices. Let's calculate the mean house price for a post code just by using the data frame and the group by function. And then we're going to replace each post code with its target mean. Just using the map function of the post code means against the data frame.

00:08:12 Speaker 2

And when you think about what's happening here in just three lines of code, it's rather powerful. But yet we just are calling these built-in functions of pandas to manipulate our data in the way that we want to get the results we achieve.

00:08:23 Speaker 2

So what have we seen in this lesson?

00:08:25 Speaker 2

We've seen how we can handle outliers, those values that go beyond the range of normal distribution. Now we saw that we could use the interquartile range as a technique for determining what values fall outside of a normal range. You can then decide how you handle those outliers. Do we drop them all together? Do we include them but cap them at windsorization technique? Interquartile ranges help us identify them. You still need to decide what to do about them.

00:08:48 Speaker 2

We saw scaling techniques, and we saw that different scaling techniques apply at different times depending on the data set and the algorithm that you've chosen. But one consistent thing that comes up with numeric data is that if you have two features that are on wildly different scales, do you want the algorithm to place more importance on those larger numeric values, or should that not really matter? 3000 square foot versus 2 bedroom. Does the actual number matter, or should we get them into the same scale and let the pattern evolve from the training pattern after that?

00:09:13 Speaker 2

We've seen that if we standardize our data, then yes, we're scaling it, but we're scaling in a specific way that we center it around 0 so that we get a mean value of 0 for that feature with a standard deviation of 1. So it doesn't bound the upper and lower values, but the values will tend to fall approximately in the range of typically about -1 to 1.

00:09:31 Speaker 2

And we've seen normalizing data and particularly appropriate for sparse data like NLP or image data and we saw that that was a rule wise operation rather than a feature or columnar operation. It row wise so that all the numeric features in that row when squared and summed equals 1. And this is to do with what we call distance vectors and those algorithms that use distance vectors that are sensitive to that.

00:09:51 Speaker 2

We have seen that where we have categorical data, text data, we can't do much with that in terms of weights and biases when we think about the sort of linear algebra under the hood and gradient descent, trying to minimize loss. But if we can turn categorical data into numeric data, then we can work with it. But we've seen if I could do ordinal encoding, but that just puts in numbers, but then it might place importance on those numbers. Or I could do 1 hot encoding, which could be adding new features, new synthesized columns. But I need to also think about high cardinality. So I need to be aware of when I might want to use.

00:10:21 Speaker 2

Beyond one hot encoding, so techniques such as target encoding.

00:10:25 Speaker 2

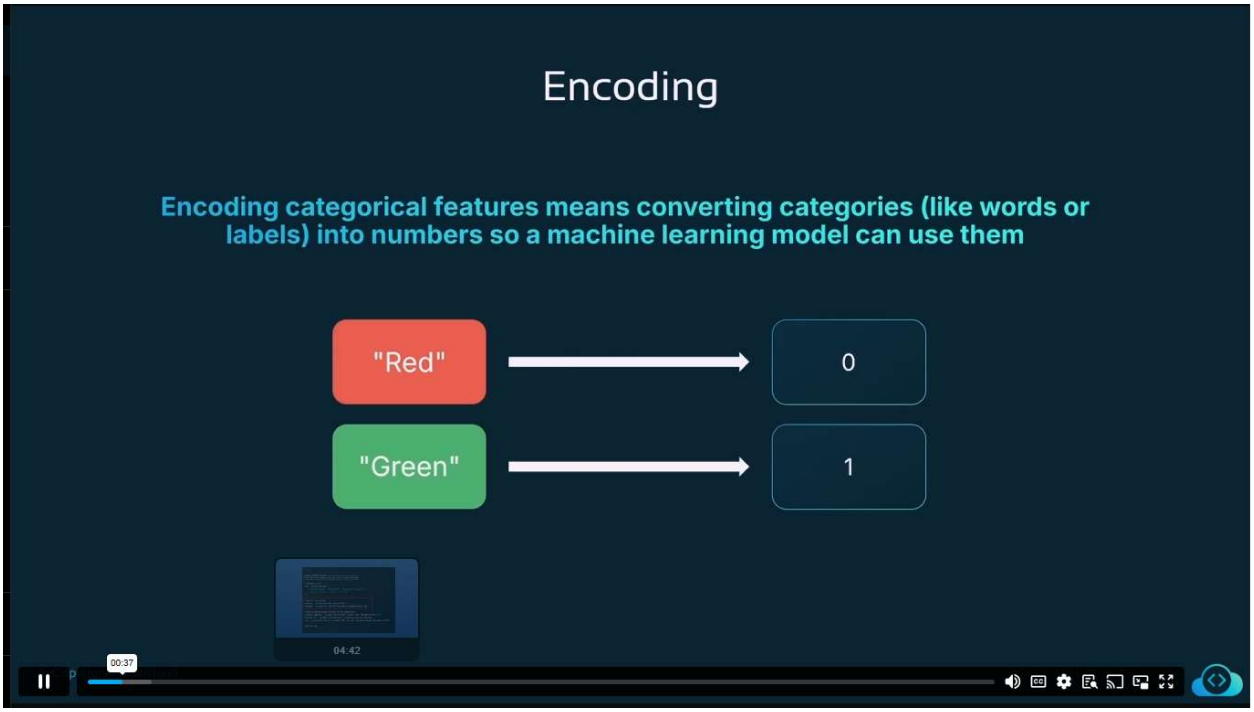
We have seen that in order to do this data transformation, we were going to be using Python packages such as pandas in order to get a data frame, and we can use numpy in addition to pandas to do additional mathematical functions. We used it in this lesson in order to clip values outside of our IQR outliers. And we can use scikit learn or sklearn as it's written in code to gain access to numerous methods that can help us perform things like min, Max, scalar, or normalization or standardization. All those standard techniques are just one function call away.

00:10:55 Speaker 2

When I use scikit learn.

00:10:57 Speaker 2

So that wraps it up here. In our next lesson, we're going to take a look at why Sagemaker, the product is seen as so mysterious and intimidating and help get you past that barrier. See you there.



Label Encoded

Neighborhood	Encoded Neighborhood
Downtown	1
Suburbs	2
Rural	3

Use the **encoded feature** (numerical representation of **neighborhood**) for training the model.

One-Hot Encoding

Neighborhood	Neighborhood_ Downtown	Neighborhood_ Suburbs	Neighborhood_ Rural
Downtown	1	0	0
Suburbs	0	1	0
Rural	0	0	1

```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Example data
data = pd.DataFrame({
    'Neighborhood': ['Downtown', 'Suburbs', 'Rural'],
    'Price': [300000, 200000, 150000]
})

# One-hot encoding
encoder = OneHotEncoder(sparse=False)
encoded = encoder.fit_transform(data[['Neighborhood']])

# Add encoded features back to the DataFrame
encoded_columns = encoder.get_feature_names_out(['Neighborhood'])
encoded_df = pd.DataFrame(encoded, columns=encoded_columns)
data = pd.concat([data, encoded_df], axis=1).drop(columns=['Neighborhood'])

print(data)
```


Target Encoding

ID	Postcode 1001	Postcode 1002	Postcode 1003
1	1	0	0
2	0	1	0
3	0	0	1

One-Hot Encoding creates a **wide dataset**,
with many columns for each unique **Postcode**.

Target Encoding

01

Group the data by the categorical variable (e.g., **Postcode**).

02

Calculate the mean (or another statistic) of the target variable (e.g., **House Price**) for each group.

03

Replace the categorical value with its corresponding mean target value.

Target Encoding

01

Dimensionality reduction

One column instead of many

02

Preserves relationships

Captures the connection between the feature and target

03

Efficient

Handles high cardinality better

Target Encoding

Original Data: House Prices and Postcodes

Postcode	House Price
A1	300,000
B2	250,000
A1	320,000
C3	150,000
B2	270,000

Target Encoding

Mean Calculation: Average House Price per Postcode

Postcode	Mean House Price for Postcode
A1	310,000
B2	260,000
A1	150,000

Target Encoding

Target Encoding: Replacing Postcodes With Mean House Price

Postcode	House Price	Target Encoded
A1	300,000	310,000
B2	250,000	260,000
A1	320,000	310,000
C3	150,000	150,000
B2	270,000	260,000

targetencoded.py

```python
import pandas as pd

# Example data
data = {
    'Postcode': ['A1', 'B2', 'A1', 'C3', 'B2'],
    'HousePrice': [300000, 250000, 320000, 150000, 270000]
}
df = pd.DataFrame(data)

# Global mean of house prices
global_mean = df['HousePrice'].mean()

# Calculate the mean house price per postcode
postcode_means = df.groupby('Postcode')['HousePrice'].mean()

# Replace each postcode with its target mean
df['TargetEncodedPostcode'] = df['Postcode'].map(postcode_means)

print(df)
```


Summary

01 **Handle outliers** and understand why they matter

02 Apply **scaling techniques** for consistent data representation

03 **Standardize data** with a mean of zero and standard deviation of one

04 **Normalize data** by scaling values between zero and one

Summary

05 **Encode categorical data** for ML models

06 Use **Python with Pandas** to manipulate multi-dimensional data

07 Use **NumPy** to clip outliers

08 Use **scikit-learn** to scale and encode data efficiently