RegressionSetup_TrainTestSplit

April 3, 2023

1 Regression Setup, Train-test Split

TASK: Import the data using Pandas and examine the shape. There are 79 feature columns plus the predictor, the sale price `SalePrice`.

[1]:
```
import pandas as pd
import numpy as np

# Import the data using the file path
URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBM-ML240EN-SkillsNetwork/labs/data/Ames_Housing_Sales.csv'

data = pd.read_csv(URL)
display(data.head())

print("shape: ", data.shape)
```

```
   1stFlrSF  2ndFlrSF  3SsnPorch Alley  BedroomAbvGr BldgType BsmtCond  \
0     856.0     854.0        0.0  None             3     1Fam       TA
1    1262.0       0.0        0.0  None             3     1Fam       TA
2     920.0     866.0        0.0  None             3     1Fam       TA
3     961.0     756.0        0.0  None             3     1Fam       Gd
4    1145.0    1053.0        0.0  None             4     1Fam       TA

  BsmtExposure  BsmtFinSF1  BsmtFinSF2  … ScreenPorch Street  TotRmsAbvGrd  \
0           No       706.0         0.0  …         0.0   Pave             8
1           Gd       978.0         0.0  …         0.0   Pave             6
2           Mn       486.0         0.0  …         0.0   Pave             6
3           No       216.0         0.0  …         0.0   Pave             7
4           Av       655.0         0.0  …         0.0   Pave             9

   TotalBsmtSF Utilities  WoodDeckSF YearBuilt YearRemodAdd YrSold SalePrice
0        856.0    AllPub         0.0      2003         2003   2008  208500.0
1       1262.0    AllPub       298.0      1976         1976   2007  181500.0
2        920.0    AllPub         0.0      2001         2002   2008  223500.0
3        756.0    AllPub         0.0      1915         1970   2006  140000.0
4       1145.0    AllPub       192.0      2000         2000   2008  250000.0
```

```
[5 rows x 80 columns]

shape:  (1379, 80)
```

[2]:
```
data.dtypes.value_counts()
```

[2]:
```
object     43
float64    21
int64      16
dtype: int64
```

TASK: A significant challenge, particularly when dealing with data that have many columns, is ensuring each column gets encoded correctly.

This is particularly true with data columns that are ordered categoricals (ordinals) vs unordered categoricals. Unordered categoricals should be one-hot encoded, however this can significantly increase the number of features and creates features that are highly correlated with each other.

Determine how many total features would be present, relative to what currently exists, if all string (object) features are one-hot encoded. Recall that the total number of one-hot encoded columns is `n-1`, where `n` is the number of categories.

[4]:
```
# Select the object (string) columns
mask = data.dtypes == object
categorical_cols = data.columns[mask]

# Determine how many extra columns would be created
num_ohc_cols = (data[categorical_cols].apply(lambda x: x.nunique()).
 ↪sort_values(ascending=False))
# No need to encode if there is only one value
small_num_ohc_cols = num_ohc_cols.loc[num_ohc_cols > 1]
# Number of one-hot columns is one less than the number of categories
small_num_ohc_cols -= 1

# This is 215 columns, assuming the original ones are dropped.
# This is quite a few extra columns!
small_num_ohc_cols.sum()
```

[4]:
```
215
```

TASK: Let's create a new data set where all of the above categorical features will be one-hot encoded. We can fit this data and see how it affects the results.

- Used the dataframe `.copy()` method to create a completely separate copy of the dataframe for one-hot encoding
- On this new dataframe, one-hot encode each of the appropriate columns and add it back to the dataframe. Be sure to drop the original column.
- For the data that are not one-hot encoded, drop the columns that are string categoricals.

For the first step, numerically encoding the string categoricals, either Scikit-learn;s `LabelEncoder` or `DictVectorizer` can be used. However, the former is probably easier since it doesn't require specifying a numerical value for each category, and we are going to one-hot encode all of the numerical values anyway. (Can you think of a time when `DictVectorizer` might be preferred?)

[6]:
```python
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Copy of the data
data_ohc = data.copy()

# The encoders
le = LabelEncoder()
ohc = OneHotEncoder()

for col in num_ohc_cols.index:

    # Integer encode the string categories
    dat = le.fit_transform(data_ohc[col]).astype(int)

    # Remove the original column from the dataframe
    data_ohc = data_ohc.drop(col, axis=1)

    # One hot encode the data (this returns a sparse array)
    new_dat = ohc.fit_transform(dat.reshape(-1,1))

    # Create unique column names
    n_cols = new_dat.shape[1]
    col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

    # Create the new dataframe
    new_df = pd.DataFrame(new_dat.toarray(),
                          index=data_ohc.index,
                          columns=col_names)

    # Append the new data to the dataframe
    data_ohc = pd.concat([data_ohc, new_df], axis=1)
```

[7]:
```python
# Column difference is as calculated above
data_ohc.shape[1] - data.shape[1]
```

[7]: 215

[8]:
```python
print("Num of features before object feature removal: ", data.shape[1])

# Remove the string columns from the dataframe
data = data.drop(num_ohc_cols.index, axis=1)

print("Num of features after object feature removal: ", data.shape[1])
```

```
Num of features before object feature removal:  80
```

```
Num of features after object feature removal:  37
```

TASK:

- Create train and test splits of both data sets. To ensure the data gets split the same way, use the same `random_state` in each of the two splits.
- For each data set, fit a basic linear regression model on the training data.
- Calculate the mean squared error on both the train and test sets for the respective models. Which model produces smaller error on the test data and why?

[9]:
```python
from sklearn.model_selection import train_test_split

y_col = 'SalePrice'

# Split the data that is not one-hot encoded
feature_cols = [x for x in data.columns if x != y_col]
X_data = data[feature_cols]
y_data = data[y_col]

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.3,␣
 ↪random_state=42)
# Split the data that is one-hot encoded
feature_cols = [x for x in data_ohc.columns if x != y_col]
X_data_ohc = data_ohc[feature_cols]
y_data_ohc = data_ohc[y_col]

X_train_ohc, X_test_ohc, y_train_ohc, y_test_ohc = train_test_split(X_data_ohc,␣
 ↪y_data_ohc,
                                                    test_size=0.3,␣
 ↪random_state=42)
```

[10]:
```python
# Compare the indices to ensure they are identical
(X_train_ohc.index == X_train.index).all()
```

[10]: True

[11]:
```python
X_train
```

[11]:

	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	\
461	630.0	0.0	0.0	1	515.0	0.0	
976	845.0	0.0	0.0	3	0.0	0.0	
1128	728.0	728.0	0.0	3	0.0	0.0	
904	561.0	668.0	0.0	2	285.0	0.0	
506	1601.0	0.0	0.0	3	1358.0	0.0	
…	…	…	…	…	…	…	
1095	855.0	601.0	0.0	3	311.0	0.0	
1130	815.0	875.0	0.0	3	0.0	0.0	
1294	1661.0	0.0	0.0	3	831.0	0.0	

```
860      742.0     742.0         0.0             3        0.0            0.0
1126    1224.0       0.0         0.0             2      883.0            0.0

      BsmtFullBath  BsmtHalfBath  BsmtUnfSF  EnclosedPorch  …  OverallCond  \
461              1             0      115.0            0.0  …            8
976              0             0        0.0            0.0  …            3
1128             0             0      728.0            0.0  …            5
904              0             0      276.0            0.0  …            6
506              1             0      223.0            0.0  …            5
…              …            …        …              …    …          …
1095             0             0      544.0            0.0  …            5
1130             0             0      815.0          330.0  …            6
1294             1             0      161.0            0.0  …            6
860              0             0      742.0            0.0  …            5
1126             1             0      341.0            0.0  …            5

      OverallQual  PoolArea  ScreenPorch  TotRmsAbvGrd  TotalBsmtSF  \
461             4       0.0          0.0             3        630.0
976             4       0.0          0.0             5          0.0
1128            6       0.0          0.0             8        728.0
904             6       0.0          0.0             5        561.0
506             8       0.0          0.0             6       1581.0
…             …       …          …            …          …
1095            6       0.0          0.0             7        855.0
1130            7       0.0          0.0             7        815.0
1294            6       0.0        178.0             8        992.0
860             6       0.0          0.0             8        742.0
1126            6       0.0          0.0             5       1224.0

      WoodDeckSF  YearBuilt  YearRemodAdd  YrSold
461          0.0       1970          2002    2009
976        186.0       1957          1957    2009
1128       100.0       2005          2005    2008
904        150.0       1980          1980    2009
506        180.0       2001          2002    2010
…           …         …            …     …
1095        26.0       1978          1978    2010
1130         0.0       1916          1950    2006
1294         0.0       1955          1996    2008
860         36.0       2005          2005    2009
1126         0.0       1999          1999    2009

[965 rows x 36 columns]
```

[12]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```python
LR = LinearRegression()

# Storage for error values
error_df = list()

# Data that have not been one-hot encoded
LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

error_df.append(pd.Series({'train': mean_squared_error(y_train, y_train_pred),
                           'test' : mean_squared_error(y_test,  y_test_pred)},
                          name='no enc'))

# Data that have been one-hot encoded
LR = LR.fit(X_train_ohc, y_train_ohc)
y_train_ohc_pred = LR.predict(X_train_ohc)
y_test_ohc_pred = LR.predict(X_test_ohc)

error_df.append(pd.Series({'train': mean_squared_error(y_train_ohc,␣
 ↪y_train_ohc_pred),
                           'test' : mean_squared_error(y_test_ohc, ␣
 ↪y_test_ohc_pred)},
                          name='one-hot enc'))

# Assemble the results
error_df = pd.concat(error_df, axis=1)
error_df
```

[12]:

	no enc	one-hot enc
train	1.131507e+09	3.177267e+08
test	1.372182e+09	2.434096e+13

Note that the error values on the one-hot encoded data are very different for the train and test data. In particular, the errors on the test data are much higher. Based on the lecture, this is because the one-hot encoded model is overfitting the data. We will learn how to deal with issues like this in the next lesson.

TASK:

For each of the data sets (one-hot encoded and not encoded):

- Scale the all the non-hot encoded values using one of the following: `StandardScaler`, `MinMaxScaler`, `MaxAbsScaler`.
- Compare the error calculated on the test sets

Be sure to calculate the skew (to decide if a transformation should be done) and fit the scaler on *ONLY* the training data, but then apply it to both the train and test data identically.

[13]:
```
# Mute the setting wtih a copy warnings
pd.options.mode.chained_assignment = None
```

[14]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler


scalers = {'standard': StandardScaler(),
           'minmax': MinMaxScaler(),
           'maxabs': MaxAbsScaler()
          }

training_test_sets = {
    'not_encoded': (X_train, y_train, X_test, y_test),
    'one_hot_encoded': (X_train_ohc, y_train_ohc, X_test_ohc, y_test_ohc)
}


# Get the list of float columns, and the float data, so that we don't scale␣
↪something we already scaled.
# We're supposed to scale the original data each time
mask = X_train.dtypes == float
float_columns = X_train.columns[mask]

# initialize model
LR = LinearRegression()

# iterate over all possible combinations and get the errors
errors = {}
for encoding_label, (_X_train, _y_train, _X_test, _y_test) in␣
↪training_test_sets.items():
    for scaler_label, scaler in scalers.items():
        trainingset = _X_train.copy()  # copy because we dont want to scale␣
↪this more than once.
        testset = _X_test.copy()
        trainingset[float_columns] = scaler.
↪fit_transform(trainingset[float_columns])
        testset[float_columns] = scaler.transform(testset[float_columns])
        LR.fit(trainingset, _y_train)
        predictions = LR.predict(testset)
        key = encoding_label + ' - ' + scaler_label + 'scaling'
        errors[key] = mean_squared_error(_y_test, predictions)

errors = pd.Series(errors)
print(errors.to_string())
print('-' * 80)
for key, error_val in errors.items():
    print(key, error_val)
```

```
not_encoded - standardscaling        1.372182e+09
not_encoded - minmaxscaling          1.372008e+09
not_encoded - maxabsscaling          1.372950e+09
one_hot_encoded - standardscaling    4.531147e+25
one_hot_encoded - minmaxscaling      8.065328e+09
one_hot_encoded - maxabsscaling      8.065328e+09
--------------------------------------------------------------------------------
not_encoded - standardscaling 1372182358.9345126
not_encoded - minmaxscaling 1372007600.2059839
not_encoded - maxabsscaling 1372949991.9617937
one_hot_encoded - standardscaling 4.531147144561635e+25
one_hot_encoded - minmaxscaling 8065327607.279107
one_hot_encoded - maxabsscaling 8065327607.290511
```

TASK:

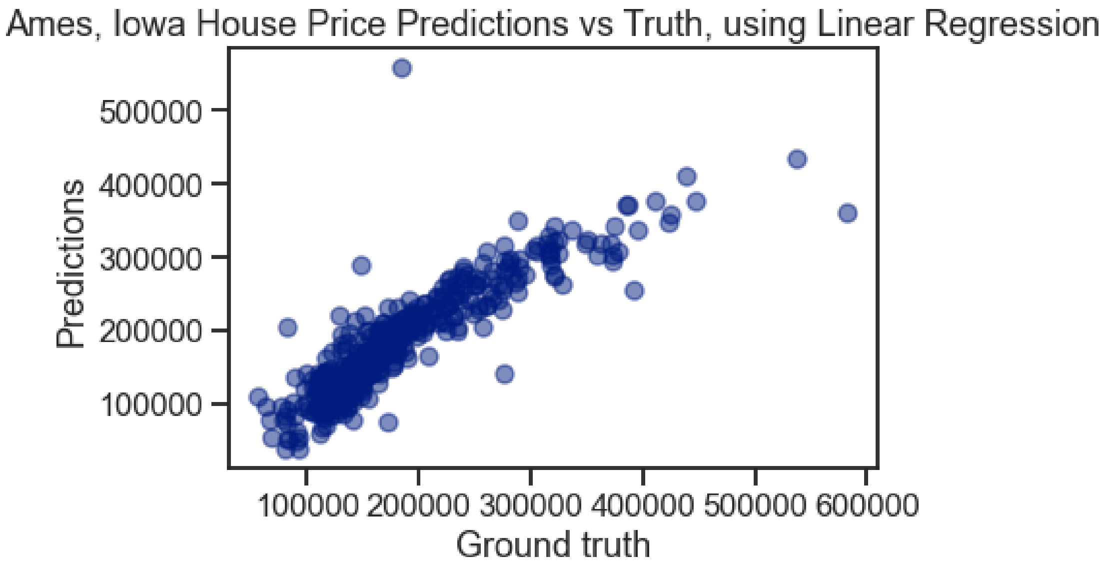
Plot predictions vs actual for one of the models.

[15]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


sns.set_context('talk')
sns.set_style('ticks')
sns.set_palette('dark')

ax = plt.axes()
# we are going to use y_test, y_test_pred
ax.scatter(y_test, y_test_pred, alpha=.5)

ax.set(xlabel='Ground truth',
       ylabel='Predictions',
       title='Ames, Iowa House Price Predictions vs Truth, using Linear␣
 ↪Regression');
```


Ames, Iowa House Price Predictions vs Truth, using Linear Regression
Predictions
500000
400000
300000
200000
100000
100000 200000 300000 400000 500000 600000
Ground truth