Week7Assignment

October 13, 2024

0.1 Part 1: PCA and Variance Threshold in a Linear Regression

1. Import the housing data as a data frame and ensure that the data is loaded properly.
2. Drop the "Id" column and any features that are missing more than 40% of their values.
3. For numerical columns, fill in any missing data with the median value.
4. For categorical columns, fill in any missing data with the most common value (mode).
5. Convert the categorical columns to dummy variables.
6. Split the data into a training and test set, where the SalePrice column is the target.
7. Run a linear regression and report the R2-value and RMSE on the test set.
8. Fit and transform the training features with a PCA so that 90% of the variance is retained (see section 9.1 in the Machine Learning with Python Cookbook).
9. How many features are in the PCA-transformed matrix?
10. Transform but DO NOT fit the test features with the same PCA.
11. Repeat step 7 with your PCA transformed data.
12. Take your original training features (from step 6) and apply a min-max scaler to them.
13. Find the min-max scaled features in your training set that have a variance above 0.1 (see Section 10.1 in the Machine Learning with Python Cookbook).
14. Transform but DO NOT fit the test features with the same steps applied in steps 11 and 12.
15. Repeat step 7 with the high variance data.
16. Summarize your findings.

0.1.1 1. Import the housing data as a data frame and ensure that the data is loaded properly.

[3]:
```
import pandas as pd
import numpy as np
```

[4]:
```
df = pd.read_csv('housing_train.csv')
```

[5]:
```
df.shape
```

[5]:
```
(1460, 81)
```

0.1.2 2. Drop the "Id" column and any features that are missing more than 40% of their values.

[7]:
```
null_count = df.isnull().sum()
pd.to_numeric(null_count)
```

[7]:
```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
                ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

[8]:
```
percent = 1460 * .4
print(percent)
```

```
584.0
```

[9]:
```
null = null_count[null_count > 584]
```

[10]:
```
null
```

[10]:
```
Alley          1369
MasVnrType      872
FireplaceQu     690
PoolQC         1453
Fence          1179
MiscFeature    1406
dtype: int64
```

[11]:
```
tobedropped =␣
↪['Alley','MasVnrType','FireplaceQu','PoolQC','Fence','MiscFeature','Id']
```

[12]:
```
df.drop(tobedropped,axis=1,inplace=True)
```

[13]:
```
df.shape
```

[13]:
```
(1460, 74)
```

0.1.3 3. For numerical columns, fill in any missing data with the median value.

0.1.4 4. For categorical columns, fill in any missing data with the most common value (mode).

[15]:
```
df.isnull().sum()
```

[15]:
```
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
Street             0
                ...
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 74, dtype: int64
```

[16]:
```
col_num = df.select_dtypes('number')
col_cat = df.select_dtypes('object')
```

[17]:
```
df[col_num.columns] = col_num.fillna(col_num.mean())
df[col_cat.columns] = col_cat.fillna(col_cat.agg(lambda x: x.mode().values[0]))
```

[18]:
```
df.isnull().sum()
```

[18]:
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
                ..
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
Length: 74, dtype: int64
```

[ ]:
```

```

0.1.5 5. Convert the categorical columns to dummy variables.

[20]:
```
df_dummies = pd.get_dummies(df[col_cat.columns])
```

[21]:
```
df.shape
```

[21]: (1460, 74)

[22]:
```
df = df.drop(df[col_cat.columns],axis = 1)
```

[23]:
```
df.shape
```

[23]: (1460, 37)

[24]:
```
df = df.join(df_dummies)
```

[25]:
```
df.shape
```

[25]: (1460, 267)

[26]:
```
df.head()
```

[26]:
```
   MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  \
0          60         65.0     8450            7            5       2003
1          20         80.0     9600            6            8       1976
2          60         68.0    11250            7            5       2001
3          70         60.0     9550            7            5       1915
4          60         84.0    14260            8            5       2000

   YearRemodAdd  MasVnrArea  BsmtFinSF1  BsmtFinSF2  …  SaleType_ConLw  \
0          2003       196.0         706           0  …           False
1          1976         0.0         978           0  …           False
2          2002       162.0         486           0  …           False
3          1970         0.0         216           0  …           False
4          2000       350.0         655           0  …           False

   SaleType_New  SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  \
0         False         False         True                  False
1         False         False         True                  False
2         False         False         True                  False
3         False         False         True                   True
4         False         False         True                  False

   SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
0                  False                 False                 False
1                  False                 False                 False
2                  False                 False                 False
3                  False                 False                 False
4                  False                 False                 False

   SaleCondition_Normal  SaleCondition_Partial
0                  True                  False
1                  True                  False
2                  True                  False
```

```
3                  False                     False
4                   True                     False

[5 rows x 267 columns]
```

```
[ ]:
```

```
[ ]:
```

0.1.6 6. Split the data into a training and test set, where the SalePrice column is the target.

```
[28]: y = df['SalePrice']
```

```
[29]: x = df.drop('SalePrice', axis=1)
```

```
[30]: # x.to_csv('x.csv')
```

```
[31]: from sklearn.model_selection import train_test_split
```

```
[32]: x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.2)
```

0.1.7 7. Run a linear regression and report the R2-value and RMSE on the test set.

```
[34]: from sklearn.linear_model import LinearRegression
```

```
[35]: model = LinearRegression().fit(x_train, y_train)
```

```
[36]: y_pred = model.predict(x_test)
```

```
[37]: from sklearn.metrics import r2_score
```

```
[38]: r2 = r2_score(y_test,y_pred)
```

```
[39]: r2
```

```
[39]: 0.9078568330885358
```

```
[40]: from sklearn.metrics import mean_squared_error
      from math import sqrt
```

```
[41]: rmse = sqrt(mean_squared_error(y_test, y_pred))
      print(rmse)
```

```
25063.582037723107
```

```
[ ]:
```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

0.1.8 8. Fit and transform the training features with a PCA so that 90% of the variance is retained (see section 9.1 in the Machine Learning with Python Cookbook).

[43]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

[44]:

```
pca = PCA(n_components=90,whiten=True)
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)
```

[ ]:

```

```

0.1.9 9. How many features are in the PCA-transformed matrix?

[46]:

```
x_train_pca.shape,x_train.shape,x_test_pca.shape,x_test.shape
```

[46]:

```
((1168, 90), (1168, 266), (292, 90), (292, 266))
```

this reduced the features from 266 to 90

[ ]:

```

```

0.1.10 10. Transform but DO NOT fit the test features with the same PCA.

oops I did this on step 8

0.1.11 11. Repeat step 7 with your PCA transformed data.

[51]:

```
model2 = LinearRegression().fit(x_train_pca, y_train)
```

[52]:

```
y_pred2 = model2.predict(x_test_pca)
```

[53]:

```
r2 = r2_score(y_test,y_pred2)
print(r2)
```

```
0.8708654260989612
```

[54]:

```
rmse = sqrt(mean_squared_error(y_test, y_pred2))
print(rmse)
```

```
29671.044793779623
```

[ ]:

[ ]:

0.1.12 12. Take your original training features (from step 6) and apply a min-max scaler to them.

[56]:
```
# import module
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
model=scaler.fit(x_train)
scaled_train_data=model.transform(x_train)
scaled_test_data=model.transform(x_test)
# print scaled features
# print(scaled_train_data)
```

[57]:
```
df_scaled_train = pd.DataFrame(scaled_train_data)
```

[ ]:

0.1.13 13. Find the min-max scaled features in your training set that have a variance above 0.1 (see Section 10.1 in the Machine Learning with Python Cookbook).

[59]:
```
from sklearn.feature_selection import VarianceThreshold
```

[ ]:

[60]:
```
    var_thres=VarianceThreshold(threshold=0.1)
    var_thres.fit(df_scaled_train)
    new_cols = var_thres.get_support()
```

[61]:
```
df_hv_train = df_scaled_train.iloc[:,new_cols]
```

[ ]:

[62]:
```
df_hv_train.head()
```

[62]:
```
          6     35   39   40   43   46   53   57   73   88  …  210  220  221  \
0  0.850000  0.25  0.0  1.0  0.0  1.0  1.0  0.0  0.0  1.0  …  0.0  0.0  1.0
1  0.316667  0.00  1.0  0.0  0.0  1.0  0.0  1.0  0.0  1.0  …  1.0  0.0  1.0
2  0.050000  0.00  1.0  0.0  0.0  1.0  1.0  0.0  0.0  1.0  …  1.0  0.0  1.0
3  0.950000  0.75  1.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0  …  0.0  1.0  0.0
4  0.950000  0.75  1.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0  …  0.0  0.0  0.0

   230  234  235  236  237  259  264
```

```
0  0.0  1.0  0.0  0.0  1.0  1.0  1.0
1  1.0  0.0  0.0  0.0  1.0  0.0  1.0
2  1.0  0.0  1.0  0.0  0.0  1.0  1.0
3  1.0  0.0  1.0  0.0  0.0  1.0  1.0
4  1.0  0.0  1.0  0.0  0.0  1.0  1.0

[5 rows x 48 columns]
```

[63]:
```
df_hv_train.columns
```

[63]:
```
Index([  6,  35,  39,  40,  43,  46,  53,  57,  73,  88, 103, 110, 113, 117,
       119, 136, 138, 142, 143, 151, 153, 158, 159, 163, 164, 169, 171, 172,
       178, 179, 184, 187, 188, 190, 193, 199, 206, 208, 210, 220, 221, 230,
       234, 235, 236, 237, 259, 264],
      dtype='int64')
```

[ ]:

[ ]:

0.1.14 14. Transform but DO NOT fit the test features with the same steps applied in steps 11 and 12.

[65]:
```
df_hv_test = var_thres.transform(scaled_test_data)
```

[ ]:

[ ]:

0.1.15 15. Repeat step 7 with the high variance data.

[67]:
```
hv_x_test = x_test.iloc[:,new_cols]
```

[68]:
```
hv_x_train = x_train.iloc[:,new_cols]
```

[69]:
```
hv_x_test.shape
```

[69]:
```
(292, 48)
```

[70]:
```
hv_x_train.shape
```

[70]:
```
(1168, 48)
```

[ ]:

[71]:
```
model = LinearRegression().fit(hv_x_train,y_train)
```

[72]:
```
y_pred = model.predict(hv_x_test)
```

[73]:
```
r2 = r2_score(y_test,y_pred)
```

[74]:
```
r2
```

[74]: 0.6046917816126323

[ ]:

[75]:
```
rmse = sqrt(mean_squared_error(y_test,y_pred))
print(rmse)
```

51913.40246205555

[ ]:

[ ]:

[ ]:

[ ]:

0.1.16 16. Summarize your findings.

r2 0.9078568330885358 rmse 25063.582037723107

pca r2 0.8708654260989612 pca rmse 29671.044793779623

hv r2 0.6046917816126323 hv rmse 51913.40246205555

R2: We see the highest r2 if the unaltered regression model. This shows that the unaltered model can explain more variability. I feel this is because there is more data overall rmse: Similarly we see that the lowes rmse is the unaltered model. Showing that this model is more accurate than the others.

[ ]:

[ ]:

0.2 Part 2: Categorical Feature Selection

1. Download the data from this link Mushroom Classification. Based on several categorical features, you will predict whether or not a mushroom is edible or poisonous.
2. Import the data as a data frame and ensure it is loaded correctly.
3. Convert the categorical features (all of them) to dummy variables.
4. Split the data into a training and test set.
5. Fit a decision tree classifier on the training set.
6. Report the accuracy and create a confusion matrix for the model prediction on the test set.
7. Create a visualization of the decision tree.
8. Use a 2-statistic selector to pick the five best features for this data (see section 10.4 of the Machine Learning with Python Cookbook).
9. Which five features were selected in step 7? Hint: Use the get_support function.

10. Repeat steps 4 and 5 with the five best features selected in step 7.
11. Summarize your findings.

0.2.1 1. Download the data from this link Mushroom Classification. Based on several categorical features, you will predict whether or not a mushroom is edible or poisonous.

[ ]:

[ ]:

[ ]:

[ ]:

0.2.2 2. Import the data as a data frame and ensure it is loaded correctly.

[81]:
```
df = pd.read_csv('mushrooms.csv')
```

[82]:
```
df.head()
```

[82]:
```
  class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0     p         x           s         n       t    p               f
1     e         x           s         y       t    a               f
2     e         b           s         w       t    l               f
3     p         x           y         w       t    p               f
4     e         x           s         g       f    n               f

  gill-spacing gill-size gill-color  … stalk-surface-below-ring  \
0            c         n          k  …                        s
1            c         b          k  …                        s
2            c         b          n  …                        s
3            c         n          n  …                        s
4            w         b          k  …                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-color  \
0                      w                      w         p          w
1                      w                      w         p          w
2                      w                      w         p          w
3                      w                      w         p          w
4                      w                      w         p          w

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g
```

```
[5 rows x 23 columns]
```

[83]:
```
df.shape
```

[83]:
```
(8124, 23)
```

[84]:
```
y = df['class']
```

[85]:
```
x = df.drop('class', axis=1)
```

0.2.3 3. Convert the categorical features (all of them) to dummy variables.

[87]:
```
df_dummies = pd.get_dummies(x)
```

[88]:
```
df_dummies.shape
```

[88]:
```
(8124, 117)
```

[89]:
```
df_dummies.columns
```

[89]:
```
Index(['cap-shape_b', 'cap-shape_c', 'cap-shape_f', 'cap-shape_k',
       'cap-shape_s', 'cap-shape_x', 'cap-surface_f', 'cap-surface_g',
       'cap-surface_s', 'cap-surface_y',
       …
       'population_s', 'population_v', 'population_y', 'habitat_d',
       'habitat_g', 'habitat_l', 'habitat_m', 'habitat_p', 'habitat_u',
       'habitat_w'],
      dtype='object', length=117)
```

[90]:
```
x = df_dummies
```

[ ]:

0.2.4 4. Split the data into a training and test set.

[92]:
```
x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.2)
```

[ ]:

[ ]:

[ ]:

0.2.5 5. Fit a decision tree classifier on the training set.

[ ]:
```

```

[94]:
```
from sklearn.tree import DecisionTreeClassifier
```

[238]:
```
dt = DecisionTreeClassifier()
```

[240]:
```
# Performing training clf_entropy.fit(X_train, y_train) return clf_entropy
```

[242]:
```
dt.fit(x_train, y_train)
```

[242]:
```
DecisionTreeClassifier()
```

[ ]:
```

```

0.2.6 6. Report the accuracy and create a confusion matrix for the model prediction on the test set.

[246]:
```
from sklearn.metrics import accuracy_score
y_pred = dt.predict(x_test)
```

[248]:
```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[248]:
```
1.0
```

[252]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[252]:
```
array([[828,   0],
       [  0, 797]], dtype=int64)
```

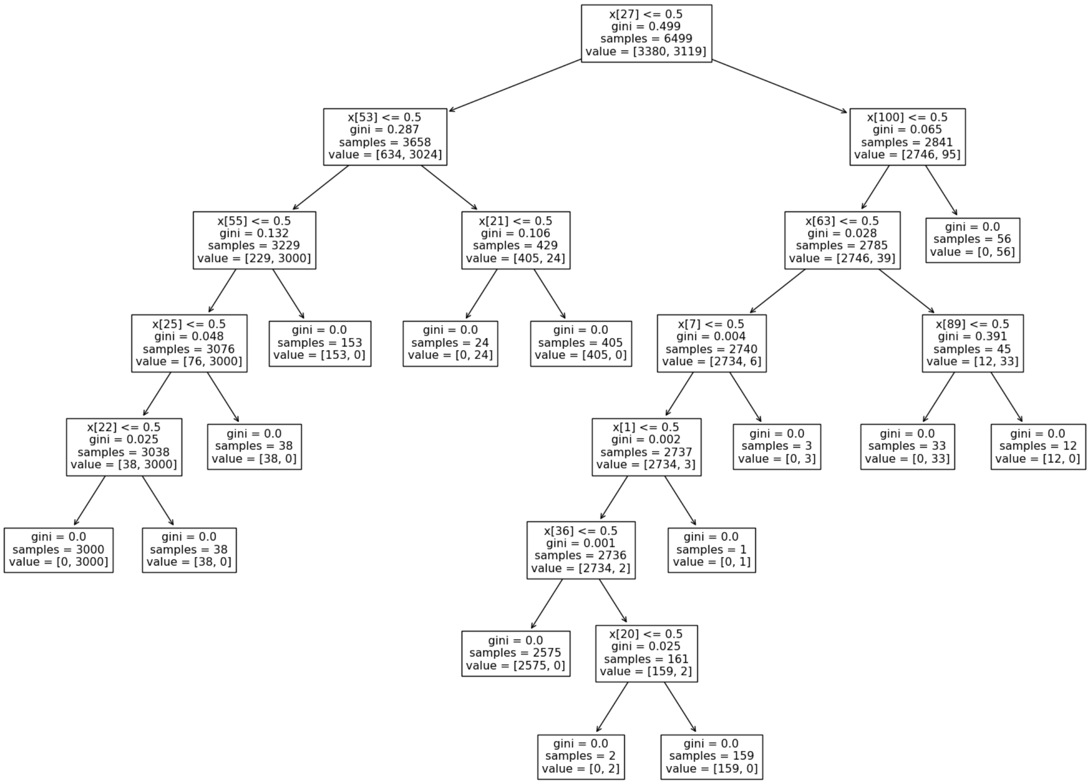
[ ]:
```

```

0.2.7 7. Create a visualization of the decision tree.

[254]:
```
import matplotlib.pyplot as plt
from sklearn import tree
```

[264]:
```
fig = plt.figure(figsize=(20,15))
tree.plot_tree(dt)
plt.show
```

[264]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[ ]:

[ ]:

[ ]:

0.2.8 8. Use a 2-statistic selector to pick the five best features for this data (see section 10.4 of the Machine Learning with Python Cookbook).

[284]:
```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
```

[286]:
```
selector = SelectKBest(chi2, k=5)
chi_x = selector.fit_transform (x,y)
```

[288]:
```
chi_x
```

[288]:
```
array([[False, False, False, False, False],
       [False, False, False, False, False],
       [False, False, False, False, False],
       …,
```

```
       [False,  True, False, False, False],
       [False, False,  True, False,  True],
       [False,  True, False, False, False]])
```

[292]:

```

```

0.2.9 9. Which five features were selected in step 7? Hint: Use the get_support function.

[294]:

```
   new_cols = selector.get_support()
```

[294]:

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
        True, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False])
```

[305]:

```
top5_x = x.iloc[:,new_cols]
```

[309]:

```
top5_x.columns
```

[309]:

```
Index(['odor_f', 'odor_n', 'gill-color_b', 'stalk-surface-above-ring_k',
       'stalk-surface-below-ring_k'],
      dtype='object')
```

[ ]:

```

```

0.2.10 10. Repeat steps 4 and 5 with the five best features selected in step 7.

[311]:

```
x_train, x_test, y_train, y_test= train_test_split(top5_x,y, test_size=0.2)
```

[313]:

```
dt = DecisionTreeClassifier()
```

[315]:

```
dt.fit(x_train, y_train)
```

[315]:

```
DecisionTreeClassifier()
```

[317]:

```
from sklearn.metrics import accuracy_score
y_pred = dt.predict(x_test)
```

[319]:
```python
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[319]:
```
0.9298461538461539
```

[321]:
```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[321]:
```
array([[848,   0],
       [114, 663]], dtype=int64)
```

[ ]:
```python

```

[ ]:
```python

```

0.2.11 11. Summarize your findings.

Looking soley at the accuracy the original decision tree appears to be more accurate. I'm a bit concerned because in the teams thread I saw someone say that both their models resulted in perfect accuarcy but I had different results. I also ran the original decison tree with entropy and that resulted in a less than perfect accuracy.

[ ]:
```python

```