Week 7: Brandon Mather DSC550-T301

Part 1: PCA and Variance Threshold in a Linear Regression

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import make_scorer, r2_score
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import VarianceThreshold
```

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

In [2]:
```
train = pd.read_csv("train.csv")
```

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

In [3]:
```
train.drop(columns=['Id'])
```

Out[3]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub
...	...	...	...	...	...	...	...	...	...
1455	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub
1456	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub
1457	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub
1458	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub
1459	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub

1460 rows × 80 columns

In [4]:
```
limitPer = len(train) * .40
```

In [5]:
```
train2 = train.dropna(thresh=limitPer, axis=1)
train2
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub
...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub

1460 rows × 77 columns

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

In [6]:

```
train3 = train2.fillna(train2.median())
train3
```

```
C:\Users\brand\AppData\Local\Temp\ipykernel_6948\751901305.py:1: FutureWarning: Dropp
ing of nuisance columns in DataFrame reductions (with 'numeric_only=None') is depreca
ted; in a future version this will raise TypeError.  Select only valid columns before
calling the reduction.
  train3 = train2.fillna(train2.median())
```

Out[6]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub
...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub

1460 rows × 77 columns

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

In [7]:

```
train3 = train3.apply(lambda x: x.fillna(x.value_counts().index[0]))
train3
```

Out[7]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	IR1	Lvl	AllPub
...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	Reg	Lvl	AllPub
1456	1457	20	RL	85.0	13175	Pave	Reg	Lvl	AllPub
1457	1458	70	RL	66.0	9042	Pave	Reg	Lvl	AllPub
1458	1459	20	RL	68.0	9717	Pave	Reg	Lvl	AllPub
1459	1460	20	RL	75.0	9937	Pave	Reg	Lvl	AllPub

1460 rows × 77 columns

5)Convert the categorical columns to dummy variables.

In [8]:
```
train3 = pd.get_dummies(train3)
```

In [9]:
```
train3
```

Out[9]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd
0	1	60	65.0	8450	7	5	2003	2003
1	2	20	80.0	9600	6	8	1976	1976
2	3	60	68.0	11250	7	5	2001	2002
3	4	70	60.0	9550	7	5	1915	1970
4	5	60	84.0	14260	8	5	2000	2000
...	...	...	...	...	...	...	...	...
1455	1456	60	62.0	7917	6	5	1999	2000
1456	1457	20	85.0	13175	6	6	1978	1988
1457	1458	70	66.0	9042	7	9	1941	2006
1458	1459	20	68.0	9717	5	6	1950	1996
1459	1460	20	75.0	9937	5	6	1965	1965

1460 rows × 277 columns

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

In [11]:
```
x = train3.SalePrice
y = train3.drop('SalePrice',axis=1)
```

In [12]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [13]:
```
x_train = x_train.values.reshape(-1, 1)
x_test = x_test.values.reshape(-1, 1)
```

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

In [14]:
```
reg = LinearRegression().fit(x_train, y_train)
```

In [15]:
```
r2 = reg.score(x_test, y_test)
r2
```

Out[15]:
```
0.0332948591155382
```

In [16]:
```
pred = reg.predict(x_test)
```

In [17]:
```
rmse = np.sqrt(mean_squared_error(y_test,pred))
rmse
```

Out[17]:
```
919.7997301596483
```

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).

In [18]:
```
pca = PCA(n_components=0.90, whiten=True)
```

In [19]:
```
x_reduced_train = pca.fit_transform(x_train)
```

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

In [20]:
```
x_reduced_train.shape[1]
```

Out[20]:
```
1
```

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

In [21]:
```
x_reduced_test = pca.transform(x_test)
```

11)Repeat step 7 with your PCA transformed data.

In [22]:
```
reg2 = LinearRegression().fit(x_reduced_train, y_train)
```

In [23]:
```
r2_2 = reg2.score(x_reduced_test, y_test)
r2_2
```

Out[23]:
```
0.03329485911553823
```

In [24]:
```
pred2 = reg2.predict(x_reduced_test)
```

In [25]:
```
rmse2 = np.sqrt(mean_squared_error(y_test,pred2))
rmse2
```

Out[25]:
```
919.7997301596483
```

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

In [26]:
```
mmscaler = MinMaxScaler()
```

In [27]:
```
min_max = mmscaler.fit_transform(x_train)
```

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).

In [28]:
```
thresholder = VarianceThreshold()
```

In [29]:
```
min_max_high_variance = thresholder.fit_transform(min_max)
```

14)Transform but DO NOT fit the test features with the same steps applied in steps 12 and 13.

In [30]:
```
min_max_test = mmscaler.transform(x_test)
```

In [31]:
```
min_max_high_variance_test = thresholder.transform(min_max_test)
```

15)Repeat step 7 with the high variance data.

In [32]:
```
reg3 = LinearRegression().fit(min_max_high_variance, y_train)
```

In [33]:
```
r2_3 = reg3.score(min_max_test, y_test)
r2_3
```

Out[33]:
```
0.033294859115538256
```

In [34]:
```
pred_3 = reg3.predict(min_max_test)
```

In [35]:
```
rmse_3 = np.sqrt(mean_squared_error(y_test,pred_3))
rmse_3
```

Out[35]:
```
919.7997301596483
```

16)Summarize your findings.

The original regression model, the PCA, and the min-max had the same r2 and rmse scores.

Part 2: Categorical Feature Selection

In [106…
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from matplotlib import pyplot as plt
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

1)Import the data as a data frame and ensure it is loaded correctly.

In [37]:
```
mushrooms = pd.read_csv("mushrooms.csv")
mushrooms
```

Out[37]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	st co abc
0	p	x	s	n	t	p	f	c	n	k	...	s	
1	e	x	s	y	t	a	f	c	b	k	...	s	
2	e	b	s	w	t	l	f	c	b	n	...	s	
3	p	x	y	w	t	p	f	c	n	n	...	s	
4	e	x	s	g	f	n	f	w	b	k	...	s	
...	...	...	...	...	...	...	...	...	...	...	...	...	
8119	e	k	s	n	f	n	a	c	b	y	...	s	
8120	e	x	s	n	f	n	a	c	b	y	...	s	
8121	e	f	s	n	f	n	a	c	b	n	...	s	
8122	p	k	y	n	f	y	f	c	n	b	...	k	
8123	e	x	s	n	f	n	a	c	b	y	...	s	

8124 rows × 23 columns

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

In [38]:
```
mushrooms = pd.get_dummies(mushrooms)
mushrooms
```

Out[38]:

	class_e	class_p	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	...
0	0	1	0	0	0	0	0	1	0	0	...
1	1	0	0	0	0	0	0	1	0	0	...
2	1	0	1	0	0	0	0	0	0	0	...
3	0	1	0	0	0	0	0	1	0	0	...
4	1	0	0	0	0	0	0	1	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
8119	1	0	0	0	0	1	0	0	0	0	...
8120	1	0	0	0	0	0	0	1	0	0	...
8121	1	0	0	0	1	0	0	0	0	0	...
8122	0	1	0	0	0	1	0	0	0	0	...
8123	1	0	0	0	0	0	0	1	0	0	...

8124 rows × 119 columns

3)Split the data into a training and test set.

In [80]:
```
X = mushrooms.class_e
Y = mushrooms.drop('class_e',axis=1)
```

In [81]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

In [82]:
```
X_train = X_train.values.reshape(-1, 1)
X_test = X_test.values.reshape(-1, 1)
```

4)Fit a decision tree classifier on the training set.

In [83]:
```
clf = tree.DecisionTreeClassifier()
```

In [84]:
```
decision_tree = clf.fit(X_train, Y_train)
```

5)Report the accuracy and create a confusion matrix for the model prediction on the test set.

In [85]:
```
predict = clf.predict(X_test)
```

In [89]:
```
metrics.accuracy_score(Y_test, predict)
```

Out[89]:
```
0.0
```

In [105…
```
confusion_matrix(Y_test, predict)
```

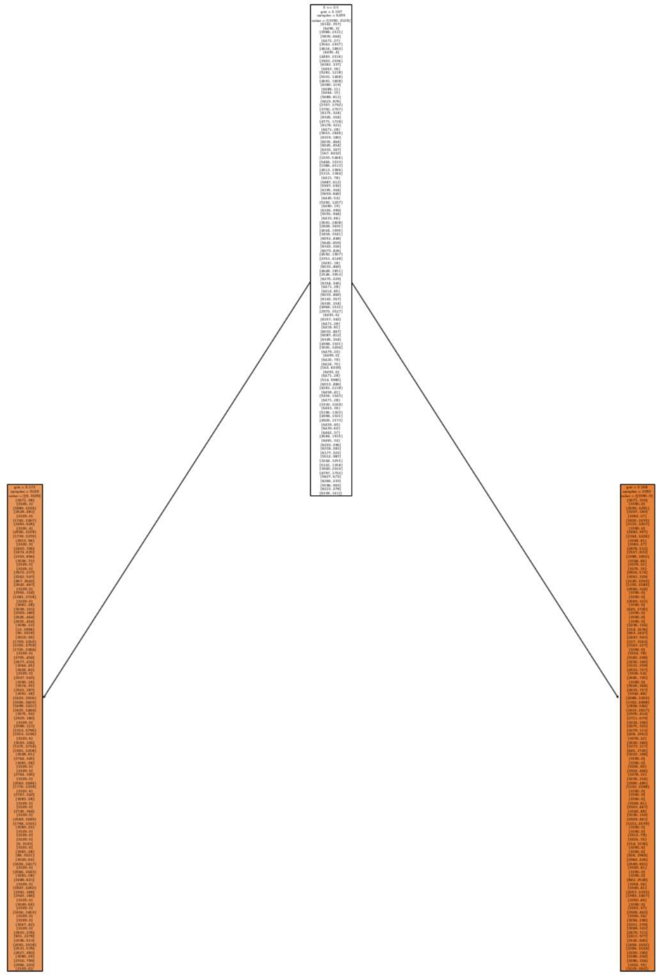
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [105], in <cell line: 1>()
----> 1 confusion_matrix(Y_test, predict)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:309, in confusi
on_matrix(y_true, y_pred, labels, sample_weight, normalize)
    307 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    308 if y_type not in ("binary", "multiclass"):
--> 309     raise ValueError("%s is not supported" % y_type)
    311 if labels is None:
    312     labels = unique_labels(y_true, y_pred)

ValueError: multilabel-indicator is not supported
```

6)Create a visualization of the decision tree.

In [100…
```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=X,
                   class_names=Y,
                   filled=True)
```



7)Use a χ2-statistic selector to pick the five best features for this data (see section 10.4 of the Machine Learning with Python Cookbook).

In [116…
```
chi2_selector = SelectKBest(chi2, k=5)
X_kbest = chi2_selector.fit_transform(X_train, Y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [116], in <cell line: 2>()
      1 chi2_selector = SelectKBest(chi2, k=5)
----> 2 X_kbest = chi2_selector.fit_transform(X_train, Y_train)

File ~\anaconda3\lib\site-packages\sklearn\base.py:855, in TransformerMixin.fit_trans
form(self, X, y, **fit_params)
    852     return self.fit(X, **fit_params).transform(X)
    853 else:
    854     # fit method of arity 2 (supervised transformation)
--> 855     return self.fit(X, y, **fit_params).transform(X)

File ~\anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.p
y:407, in _BaseFilter.fit(self, X, y)
    401 if not callable(self.score_func):
    402     raise TypeError(
    403         "The score function should be a callable, %s (%s) was passed."
    404         % (self.score_func, type(self.score_func))
    405     )
--> 407 self._check_params(X, y)
    408 score_func_ret = self.score_func(X, y)
    409 if isinstance(score_func_ret, (list, tuple)):

File ~\anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.p
y:604, in SelectKBest._check_params(self, X, y)
    602 def _check_params(self, X, y):
    603     if not (self.k == "all" or 0 <= self.k <= X.shape[1]):
--> 604         raise ValueError(
    605             "k should be >=0, <= n_features = %d; got %r. "
    606             "Use k='all' to return all features." % (X.shape[1], self.k)
    607         )

ValueError: k should be >=0, <= n_features = 1; got 5. Use k='all' to return all feat
ures.
```

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

In [ ]:

9)Repeat steps 4 and 5 with the five best features selected in step 7.

In [ ]:

10)Summarize your findings.

In [ ]: