Brian_Reppeto_DSC550_Week_7

April 28, 2024

0.0.1 DSC 550 Week :

Activity 7.2

Author: Brian Reppeto 4/22/2024

[36]:
```
# import libraries

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 mean_squared_error, r2_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2
from sklearn import tree
```

[37]:
```
# load the dataset

data=pd.read_csv('train.csv')
```

[38]:
```
# head the df

data.head(15)
```

[38]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	\
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	
5	6	50	RL	85.0	14115	Pave	NaN	IR1	
6	7	20	RL	75.0	10084	Pave	NaN	Reg	
7	8	60	RL	NaN	10382	Pave	NaN	IR1	
8	9	50	RM	51.0	6120	Pave	NaN	Reg	

```
9   10          190       RL         50.0     7420   Pave   NaN      Reg
10  11           20       RL         70.0    11200   Pave   NaN      Reg
11  12           60       RL         85.0    11924   Pave   NaN      IR1
12  13           20       RL          NaN    12968   Pave   NaN      IR2
13  14           20       RL         91.0    10652   Pave   NaN      IR1
14  15           20       RL          NaN    10920   Pave   NaN      IR1

   LandContour Utilities  … PoolArea PoolQC  Fence MiscFeature MiscVal  \
0          Lvl    AllPub  …        0    NaN    NaN         NaN       0
1          Lvl    AllPub  …        0    NaN    NaN         NaN       0
2          Lvl    AllPub  …        0    NaN    NaN         NaN       0
3          Lvl    AllPub  …        0    NaN    NaN         NaN       0
4          Lvl    AllPub  …        0    NaN    NaN         NaN       0
5          Lvl    AllPub  …        0    NaN  MnPrv        Shed     700
6          Lvl    AllPub  …        0    NaN    NaN         NaN       0
7          Lvl    AllPub  …        0    NaN    NaN        Shed     350
8          Lvl    AllPub  …        0    NaN    NaN         NaN       0
9          Lvl    AllPub  …        0    NaN    NaN         NaN       0
10         Lvl    AllPub  …        0    NaN    NaN         NaN       0
11         Lvl    AllPub  …        0    NaN    NaN         NaN       0
12         Lvl    AllPub  …        0    NaN    NaN         NaN       0
13         Lvl    AllPub  …        0    NaN    NaN         NaN       0
14         Lvl    AllPub  …        0    NaN   GdWo         NaN       0

   MoSold YrSold  SaleType  SaleCondition  SalePrice
0       2   2008        WD         Normal     208500
1       5   2007        WD         Normal     181500
2       9   2008        WD         Normal     223500
3       2   2006        WD        Abnorml     140000
4      12   2008        WD         Normal     250000
5      10   2009        WD         Normal     143000
6       8   2007        WD         Normal     307000
7      11   2009        WD         Normal     200000
8       4   2008        WD        Abnorml     129900
9       1   2008        WD         Normal     118000
10      2   2008        WD         Normal     129500
11      7   2006       New        Partial     345000
12      9   2008        WD         Normal     144000
13      8   2007       New        Partial     279500
14      5   2008        WD         Normal     157000

[15 rows x 81 columns]
```

[39]:
```
# drop the Id column and columns with more than 40% missing values

data.drop('Id', axis=1, inplace=True)
threshold=len(data) * 0.6
```

```
data.dropna(thresh=threshold, axis=1, inplace=True)
```

[40]:
```
# shape data

data.shape
```

[40]: (1460, 74)

[41]:
```
# fill missing values for numerical columns with the median

numerical_cols=data.select_dtypes(include=[np.number]).columns
data[numerical_cols]=data[numerical_cols].fillna(data[numerical_cols].median())
```

[43]:
```
# fill missing values for categorical columns with the mode

categorical_cols=data.select_dtypes(include=['object']).columns
for col in categorical_cols:
    data[col]=data[col].fillna(data[col].mode()[0])
```

[44]:
```
# convert categorical columns to dummy variables

data=pd.get_dummies(data, columns=categorical_cols)
```

[45]:
```
# split the data into training and test sets

X=data.drop('SalePrice', axis=1)
y=data['SalePrice']
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[46]:
```
# linear regression and evaluate

model=LinearRegression()
model.fit(X_train, y_train)
y_pred=model.predict(X_test)
print("Linear Regression R2-value:", r2_score(y_test, y_pred))
print("Linear Regression RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
Linear Regression R2-value: 0.8851582784349399
Linear Regression RMSE: 29679.51257347344
```

[47]:
```
# PCA on training data

pca=PCA(n_components=0.9)
X_train_pca=pca.fit_transform(X_train)
```

[48]:
```
# number of features in PCA-transformed matrix
```

```
print("Number of PCA features:", X_train_pca.shape[1])
```

```
Number of PCA features: 1
```

[49]:
```
# transform test data with the same PCA

X_test_pca = pca.transform(X_test)
```

[50]:
```
# repeat regression with PCA-transformed data

model.fit(X_train_pca, y_train)
y_pred_pca=model.predict(X_test_pca)
print("PCA Linear Regression R2-value:", r2_score(y_test, y_pred_pca))
print("PCA Linear Regression RMSE:", np.sqrt(mean_squared_error(y_test,␣
 ↪y_pred_pca)))
```

```
PCA Linear Regression R2-value: 0.06348978225830182
PCA Linear Regression RMSE: 84754.58020922921
```

[51]:
```
# apply Min-Max Scaler to original training features

scaler=MinMaxScaler()
X_train_scaled=scaler.fit_transform(X_train)
```

[52]:
```
# find scaled features with variance above 0.1

variances=np.var(X_train_scaled, axis=0)
high_variance_features=variances > 0.1
X_train_high_var=X_train_scaled[:, high_variance_features]
```

[53]:
```
# transform test data with the same scaling and selection

X_test_scaled=scaler.transform(X_test)
X_test_high_var=X_test_scaled[:, high_variance_features]
```

[54]:
```
# repeat regression with high variance data

model.fit(X_train_high_var, y_train)
y_pred_high_var=model.predict(X_test_high_var)
print("High Variance Linear Regression R2-value:", r2_score(y_test,␣
 ↪y_pred_high_var))
print("High Variance Linear Regression RMSE:", np.
 ↪sqrt(mean_squared_error(y_test, y_pred_high_var)))
```

```
High Variance Linear Regression R2-value: 0.6640119762594767
High Variance Linear Regression RMSE: 50765.51727105659
```

1 Summary of Housing Data Analysis

1.1 Linear Regression with Original Features

- **R² Value**: Linear Regression R2-value: 0.8851582784349399
- **Root Mean Squared Error (RMSE)**: Linear Regression RMSE: 29679.51257347344

1.2 Linear Regression with PCA-transformed Features

- **Number of Features After PCA**: 1
- **R² Value**: PCA Linear Regression R2-value: 0.06348978225830182
- **Root Mean Squared Error (RMSE)**: PCA Linear Regression RMSE: 84754.58020922921

1.3 Linear Regression with High Variance Features

- **R² Value**: High Variance Linear Regression R2-value: 0.6640119762594767
- **Root Mean Squared Error (RMSE)**: High Variance Linear Regression RMSE: 50765.51727105659

1.3.1 Part 2: Categorical Feature Selection

[56]:
```
# import mushroom data

mush_df=pd.read_csv('mushrooms.csv')
```

[57]:
```
# head df

mush_df.head(15)
```

[57]:

	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	
5	e	x	y	y	t	a	f	
6	e	b	s	w	t	a	f	
7	e	b	y	w	t	l	f	
8	p	x	y	w	t	p	f	
9	e	b	s	y	t	a	f	
10	e	x	y	y	t	l	f	
11	e	x	y	y	t	a	f	
12	e	b	s	y	t	a	f	
13	p	x	y	w	t	p	f	
14	e	x	f	n	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
5            c         b          n  …                      s
6            c         b          g  …                      s
7            c         b          n  …                      s
8            c         n          p  …                      s
9            c         b          g  …                      s
10           c         b          g  …                      s
11           c         b          n  …                      s
12           c         b          w  …                      s
13           c         n          k  …                      s
14           w         b          n  …                      f
```

```
   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
5                       w                       w         p          w
6                       w                       w         p          w
7                       w                       w         p          w
8                       w                       w         p          w
9                       w                       w         p          w
10                      w                       w         p          w
11                      w                       w         p          w
12                      w                       w         p          w
13                      w                       w         p          w
14                      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            o         p                 k          n       g
6            o         p                 k          n       m
7            o         p                 n          s       m
8            o         p                 k          v       g
9            o         p                 k          s       m
10           o         p                 n          n       g
11           o         p                 k          s       m
12           o         p                 n          s       g
13           o         p                 n          v       u
14           o         e                 k          a       g
```

```
[15 rows x 23 columns]
```

[58]:
```
# convert all categorical features to dummy variables

mush_df=pd.get_dummies(mush_df, columns=[col for col in mush_df.columns if col !
 ↪= 'class'])
target=mush_df['class'].map({'e': 0, 'p': 1})  # encoding target: e (edible) as␣
 ↪0, p (poisonous) as 1
features=mush_df.drop('class', axis=1)
```

[59]:
```
# split the data into a training and test set


X_train, X_test, y_train, y_test=train_test_split(features, target, test_size=0.
 ↪3, random_state=42)
```

[60]:
```
# fit a decision tree classifier on the training set

model=DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

[60]:
```
DecisionTreeClassifier(random_state=42)
```

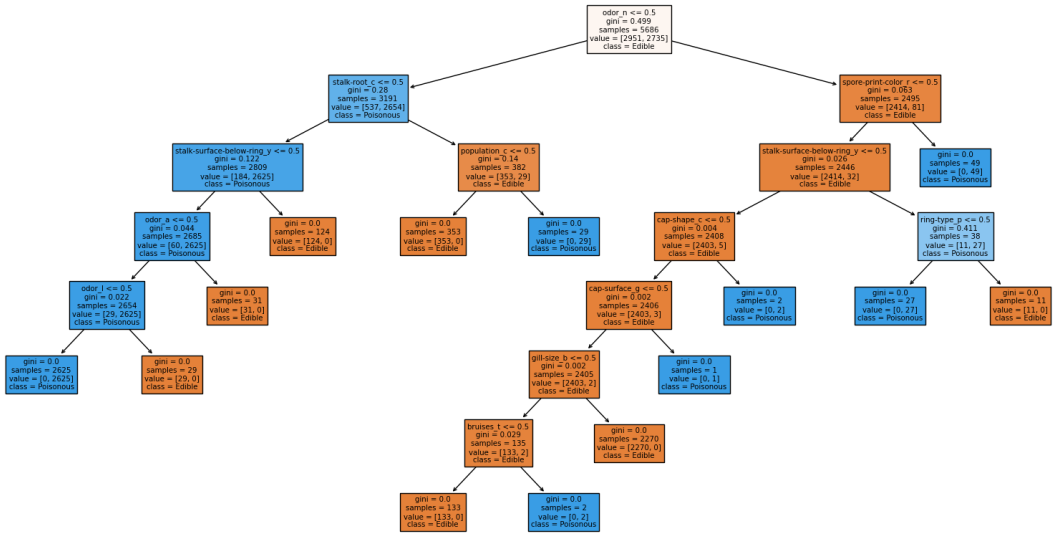
[61]:
```
#  report the accuracy and create a confusion matrix for the model prediction␣
 ↪on the test set

y_pred=model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.0
Confusion Matrix:
 [[1257    0]
 [   0 1181]]
```

[62]:
```
# create a visualization of the decision tree

plt.figure(figsize=(20,10))
tree.plot_tree(model, filled=True, feature_names=features.columns.tolist(),␣
 ↪class_names=['Edible', 'Poisonous'])
plt.show()
```



[63]:
```
# use a x2-statistic selector to pick the five best features for this data

chi2_selector=SelectKBest(chi2, k=5)
X_kbest=chi2_selector.fit_transform(features, target)
```

[64]:
```
# five features selected

selected_features=features.columns[chi2_selector.get_support()]
print("Selected features:", selected_features)
```

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

[65]:
```
# five best features selected

X_train_best, X_test_best=train_test_split(X_kbest, test_size=0.3,␣
 ↪random_state=42)
model.fit(X_train_best, y_train)
y_pred_best=model.predict(X_test_best)
print("Accuracy with best features:", accuracy_score(y_test, y_pred_best))
print("Confusion Matrix with best features:\n", confusion_matrix(y_test,␣
 ↪y_pred_best))
```

```
Accuracy with best features: 0.9249384741591469
Confusion Matrix with best features:
 [[1257    0]
 [ 183  998]]
```

2 Summary of findings

Summary of findings: The initial model using all features had an accuracy of 1.0, while using the top 5 features selected by chi-squared the accuracy was 0.9249384741591469.

[ ]: