Feature Selection

November 22, 2021

Univariate Selection

- Pearson's Correlation Coefficient: f_regression()
- ANOVA: f_classif()
  from sklearn.feature_selection import SelectKBest, f_classif
  selector = SelectKBest(f_classif, k=5)
  selector.fit(features_df, target)
- Chi-Squared: chi2()
  from sklearn.feature_selection import SelectKBest, chi2
  selector = SelectKBest(chi2, k=5)
  selector.fit(features_df, target)
- Mutual Information: mutual_info_classif() and mutual_info_regression()
- Chi-Squared: chi2()
  from sklearn.feature_selection import SelectKBest, mutual_info_classif
  selector = SelectKBest(mutual_info_classif, k=5)
  selector.fit(features_df, target)
  All the listed functions are found in the scikit-learn library

Feature selection

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif, f_regression,  chi2, mutual_info_classif

best_features = SelectKBest(score_func=f_classif, k=4)
fit = best_features.fit(Xd_train_knn,y_train_knn)

df_scores = pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(Xd_train_knn.columns)

# concatenate dataframes
feature_scores = pd.concat([df_columns, df_scores],axis=1)
feature_scores.columns = ['Feature_Name','Score']  # name output columns
print(feature_scores.nlargest(4,'Score'))  # print all 4 features
```

```
    Feature_Name     Score
2        Student  3.035842
0            Age  3.000000
3  Credit rating  0.977208
1         Income  0.567568
```

```
best_features = SelectKBest(score_func=chi2, k=4)
fit = best_features.fit(Xd_train_knn,y_train_knn)

df_scores = pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(Xd_train_knn.columns)

# concatenate dataframes
feature_scores = pd.concat([df_columns, df_scores],axis=1)
feature_scores.columns = ['Feature_Name','Score']  # name output columns
print(feature_scores.nlargest(4,'Score'))  # print all 4 features
```

```
    Feature_Name     Score
0            Age  1.028571
3  Credit rating  0.612500
2        Student  0.378125
1         Income  0.150000
```

```
best_features = SelectKBest(score_func=mutual_info_classif, k=4)
fit = best_features.fit(Xd_train_knn,y_train_knn)

df_scores = pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(Xd_train_knn.columns)

# concatenate dataframes
feature_scores = pd.concat([df_columns, df_scores],axis=1)
feature_scores.columns = ['Feature_Name','Score']  # name output columns
print(feature_scores.nlargest(4,'Score'))  # print all 4 features
```

```
    Feature_Name     Score
2        Student  0.375926
0            Age  0.275265
3  Credit rating  0.040873
1         Income  0.000000
```

Restaurant data

choice	bar	day	hungry	patron	price	rain	booking	type	time	**wait**
T	F	F	T	some	$$$	F	T	french	0	yes
T	F	F	T	full	$	F	F	thai	40	no
T	T	F	F	some	$	F	F	swiss	0	yes
T	F	T	T	full	$	F	F	thai	20	yes
T	F	T	F	full	$$$	F	T	french	60	no
F	T	F	T	some	$$	T	F	italian	0	yes
F	T	F	F	none	$	T	F	swiss	20	no
F	F	F	T	some	$$	T	T	thai	0	yes
F	T	T	F	full	$	T	F	swiss	60	no
T	T	T	T	full	$$$	F	T	italian	20	no
F	F	F	F	none	$	F	F	thai	0	no
T	T	T	T	full	$	F	F	swiss	40	yes

```
import pandas as pd
pd.set_option('display.max_colwidth', None)
Restaurant = pd.read_csv('/Users/catherine/Desktop/NLP/MachineLearning/MachineLearning2021/Restaurant.csv')
print(Restaurant.head(25))
```

```
   choice bar day hungry patron price rain booking     type  time class
0       T   F   F      T   some   $$$    F       T   french     0   yes
1       T   F   F      T   full    $     F       F     thai    40    no
2       T   T   F      F   some    $     F       F    swiss     0   yes
3       T   F   T      T   full    $     F       F     thai    20   yes
4       T   F   T      F   full   $$$    F       T   french    60    no
5       F   T   F      T   some    $$    T       F  italian     0   yes
6       F   T   F      F   none    $     T       F    swiss    20    no
7       F   F   F      T   some    $$    T       T     thai     0   yes
8       F   T   T      F   full    $     T       F    swiss    60    no
9       T   T   T      T   full   $$$    F       T  italian    20    no
10      F   F   F      F   none    $     F       F     thai     0    no
11      T   T   T      T   full    $     F       F    swiss    40   yes
```

```
Restaurant.describe(include='all')
```

	choice	bar	day	hungry	patron	price	rain	booking	type	time	class
count	12	12	12	12	12	12	12	12	12	12.000000	12
unique	2	2	2	2	3	3	2	2	4	NaN	2
top	T	T	F	T	full	$	F	F	thai	NaN	no
freq	7	6	7	7	6	7	8	8	4	NaN	6
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	21.666667	NaN
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.290003	NaN
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	NaN
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	NaN
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.000000	NaN
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.000000	NaN
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	60.000000	NaN

Split data

```
x_train = Restaurant[["choice", "bar", "day", "hungry", "patron", "price", "rain", "booking",
le = preprocessing.LabelEncoder()
#converts to 0 and 1
x_train = pd.DataFrame(columns=x_train.columns, data=le.fit_transform(x_train.values.flatten()
x_train["time"]=Restaurant["time"]

y = le.fit(Restaurant["class"])
y = le.transform(Restaurant["class"])

Xd_train, Xd_test, y_train, y_test = train_test_split(x_train, y, test_size=0.35)

print(Xd_train)
print("y_train = ",y_train, "\n")

print(Xd_test)
print("y_train = ",y_test)
```

```
    choice  bar  day  hungry  patron  price  rain  booking  type  time
3        4    3    4       4       6      0     3        3    11    20
8        3    4    4       3       6      0     4        3    10    60
5        3    4    3       4       9      1     4        3     7     0
9        4    4    4       4       6      2     3        4     7    20
6        3    4    3       3       8      0     4        3    10    20
0        4    3    3       4       9      2     3        4     5     0
10       3    3    3       3       8      0     3        3    11     0
y_train =  [1 0 1 0 0 1 0]

    choice  bar  day  hungry  patron  price  rain  booking  type  time
2        4    4    3       3       9      0     3        3    10     0
7        3    3    3       4       9      1     4        4    11     0
1        4    3    3       4       6      0     3        3    11    40
11       4    4    4       4       6      0     3        3    10    40
4        4    3    4       3       6      2     3        4     5    60
y_train =  [1 1 0 1 0]
```

GaussianNB

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
clf=GaussianNB()

clf.fit(Xd_train, y_train)

y_pred = clf.predict(Xd_test)

NB_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("NB_Accuracy = ", NB_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
y_test =  [1 1 0 1 0]
y_pred =  [1 0 0 1 0]

NB_Accuracy =  0.8

confusion_matrix
 [[2 0]
 [1 2]]
```

DecisionTreeClassifier

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
clf=DecisionTreeClassifier()

clf.fit(Xd_train, y_train)

y_pred = clf.predict(Xd_test)

DT_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("DT_Accuracy = ", DT_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
y_test =  [1 1 0 1 0]
y_pred =  [1 1 0 1 1]

DT_Accuracy =  0.8

confusion_matrix
 [[1 1]
 [0 3]]
```

KNeighborsClassifier

```
from sklearn.metrics import accuracy_score, classification_report, confusion_ma
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier()

clf.fit(Xd_train, y_train)

y_pred = clf.predict(Xd_test)

knn_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("knn_Accuracy = ", knn_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
y_test =  [1 1 0 1 0]
y_pred =  [0 0 0 0 0]

knn_Accuracy =  0.4

confusion_matrix
 [[2 0]
 [3 0]]
```

GaussianNB, chi2

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
clf=GaussianNB()

selector = SelectKBest(score_func=chi2, k=5)
fit = selector.fit(Xd_train,y_train)
train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)


clf.fit(train_selected_features, y_train)

y_pred = clf.predict(test_selected_features)

knn_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("knn_Accuracy = ", knn_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
y_test =  [1 1 0 1 0]
y_pred =  [1 0 0 1 0]

knn_Accuracy =  0.8

confusion_matrix
 [[2 0]
 [1 2]]
```

DecisionTreeClassifier, chi2

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
clf=DecisionTreeClassifier()

selector = SelectKBest(score_func=chi2, k=5)
fit = selector.fit(Xd_train,y_train)
train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)
#

clf.fit(train_selected_features, y_train)

y_pred = clf.predict(test_selected_features)

knn_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("knn_Accuracy = ", knn_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
y_test =  [1 1 0 1 0]
y_pred =  [0 0 0 0 0]

knn_Accuracy =  0.4

confusion_matrix
 [[2 0]
 [3 0]]
```

KNeighborsClassifier, chi2

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif, mutual_info_classif
clf=KNeighborsClassifier()

selector = SelectKBest(score_func=chi2, k=5)
fit = selector.fit(Xd_train,y_train)
train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)


clf.fit(train_selected_features, y_train)

y_pred = clf.predict(test_selected_features)

knn_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("knn_Accuracy = ", knn_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
y_test =  [1 1 0 1 0]
y_pred =  [0 0 0 0 0]

knn_Accuracy =  0.4

confusion_matrix
 [[2 0]
 [3 0]]
```

Recursive Feature Elimination

- The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain.
- It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.
- RFE class is in the scikit-learn

DecisionTreeClassifier, RFE

```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()

rfe = RFE(model, 5)
fit = rfe.fit(Xd_train, y_train)
print("Num Features: %d" % fit.n_features_)

print("Feature Ranking: %s" % fit.ranking_)

train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)


model.fit(train_selected_features, y_train)


y_pred = model.predict(test_selected_features)

DT_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("DT_Accuracy = ", DT_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
Num Features: 5
Feature Ranking: [6 5 2 1 4 1 3 1 1 1]
y_test =  [1 1 0 1 0]
y_pred =  [0 1 1 1 0]

DT_Accuracy =  0.6

confusion_matrix
 [[1 1]
 [1 2]]
```

Principal Component Analysis

- Principal Component Analysis (or PCA) is called a data reduction technique.
- Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form.
- In Principal Component Analysis (or PCA) you can choose the number of dimensions or principal component in the transformed result.

DecisionTreeClassifier, PCA

```
# Feature Extraction with PCA
import numpy
from sklearn.decomposition import PCA
clf = DecisionTreeClassifier()
# load data

# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(Xd_train)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_,  " \n")
print(fit.components_)

train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)


clf.fit(train_selected_features, y_train)


y_pred = clf.predict(test_selected_features)

DT_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("DT_Accuracy = ", DT_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
Explained Variance: [0.9833361  0.01241221 0.00328858]

[[-0.0041851   0.01138649  0.01768714 -0.01044247 -0.04793701 -0.01681897
   0.01036193 -0.0052808   0.0421441   0.99746616]
 [-0.09318311 -0.06683098 -0.01325886 -0.11989201 -0.09400822 -0.34389188
  -0.00205653 -0.1585915   0.90418328 -0.04998568]
 [ 0.34867307 -0.15609466  0.28821736  0.21797721 -0.72408708  0.20921983
  -0.343009    0.16995785  0.08917818 -0.03015963]]
y_test =  [1 1 0 1 0]
y_pred =  [1 1 0 0 0]

DT_Accuracy =  0.8

confusion_matrix
 [[2 0]
 [1 2]]
```

KNeighborsClassifier, PCA

```
# Feature Extraction with PCA
import numpy
from sklearn.decomposition import PCA
clf=KNeighborsClassifier()
# load data

# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(Xd_train)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_,  " \n")
print(fit.components_)

train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)


clf.fit(train_selected_features, y_train)


y_pred = clf.predict(test_selected_features)

KNN_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("KNN_Accuracy = ", KNN_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
Explained Variance: [0.9833361  0.01241221 0.00328858]

[[-0.0041851   0.01138649  0.01768714 -0.01044247 -0.04793701 -0.01681897
   0.01036193 -0.0052808   0.0421441   0.99746616]
 [-0.09318311 -0.06683098 -0.01325886 -0.11989201 -0.09400822 -0.34389188
  -0.00205653 -0.1585915   0.90418328 -0.04998568]
 [ 0.34867307 -0.15609466  0.28821736  0.21797721 -0.72408708  0.20921983
  -0.343009    0.16995785  0.08917818 -0.03015963]]
y_test =  [1 1 0 1 0]
y_pred =  [1 1 0 0 0]

KNN_Accuracy =  0.8

confusion_matrix
 [[2 0]
 [1 2]]
```

GaussianNB, PCA

```
# Feature Extraction with PCA
import numpy
from sklearn.decomposition import PCA
clf=GaussianNB()
# load data

# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(Xd_train)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_,  " \n")
print(fit.components_)

train_selected_features = fit.transform(Xd_train)
test_selected_features = fit.transform(Xd_test)


clf.fit(train_selected_features, y_train)


y_pred = clf.predict(test_selected_features)

GN_Accuracy = accuracy_score(y_test, y_pred)

print("y_test = ",y_test)
print("y_pred = ",y_pred,"\n")
print("GN_Accuracy = ", GN_Accuracy, "\n")
print("confusion_matrix \n", confusion_matrix(y_test, y_pred))
```

```
Explained Variance: [0.9833361  0.01241221 0.00328858]

[[-0.0041851   0.01138649  0.01768714 -0.01044247 -0.04793701 -0.01681897
   0.01036193 -0.0052808   0.0421441   0.99746616]
 [-0.09318311 -0.06683098 -0.01325886 -0.11989201 -0.09400822 -0.34389188
  -0.00205653 -0.1585915   0.90418328 -0.04998568]
 [ 0.34867307 -0.15609466  0.28821736  0.21797721 -0.72408708  0.20921983
  -0.343009    0.16995785  0.08917818 -0.03015963]]
y_test =  [1 1 0 1 0]
y_pred =  [1 1 0 0 0]

GN_Accuracy =  0.8
```

Classifiers	Features				
	All = 10	chi2=5	RFE=5	PCA=3	
DecisionTreeClassifier	0.8	0.4	0.6	0.8	
KNeighborsClassifier	0.4	0.4	-	0.8	
GaussianNB	0.8	0.8	-	0.8	

Feature Importance

- Decision trees can be used to estimate the importance of features.
- An importance score for each attribute where the larger score indicates the more important of the attributes

```
# Feature Importance with Decision Trees Classifier
model = DecisionTreeClassifier()

model.fit(Xd_train, y_train)

print(" importance score = \n", model.feature_importances_)
```

```
 importance score =
 [0.         0.14583333 0.         0.5625     0.         0.
 0.         0.         0.         0.29166667]
```