

10/06/2020

→ Area Under the ROC curve

[DATACAMP]

→ Large area under the ROC curve = better model

→ AUC in scikit learn

from sklearn import metrics import roc_auc_score

logreg = LogisticRegression()

X_train, X_test, y_train, y_test = train_test_split
(X, y, test_size=0.4, random_state=42)

logreg.fit(X_train, y_train)

y_pred_prob = logreg.predict_proba(X_test)[:, 1]

roc_auc_score(y_test, y_pred_prob)

↳ output: 0.994766

→ AUC using cross-validation

from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(logreg, X, y, cv=5, scoring='roc_auc')

print(cv_scores)

↳ output

[0.99673203 0.99183607 0.99593796 1.
0.96140652]

± Hyperparameter Tuning

- Linear Regression: choosing Parameters
- Ridge/Lasso Regression: choosing alpha
- k-Nearest Neighbor: choosing n_neighbr
- Parameters like α & k : Hyperparameters
- Hyperparameters cannot be learned by fitting the model.

± choosing the correct hyperparameter

- Try a bunch of different hyperparameter values.
- Fit all of them separately.
- see how well each performs.
- choosing the best performing one.
- It is essential to use cross-validation

± Grid Search Cross-validation

0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675
	0.1	0.2	0.3	0.4

Alpha

- from GridSearchCV cross-validation we have found that
 - at alpha 0.2 & C = 0.3 we will be having higher accuracy.

* Grid search CV in scikit learn

```
from sklearn.model_selection import
GridSearchCV
param_grid = {'n_neighbors': np.arange(1,50)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid,
cv=5)
```

knn_cv.fit(X, y)

knn_cv.best_params_

↳ output

{ n_neighbors: 12 }

knn_cv.best_score_

↳ output

0.93216168717

NOTE :-

Just like above i have used GridSearchCV for KNN , I can implement it for logistic Regression too.

• Hold-out set for final evaluation

Hold-out set reasoning

- How well the model perform on never seen data?
- Using all data for cross-validation is not ideal.
- Split data into training & hold-out set at the beginning.
- Perform GridSearch cross-validation on training set
- Choose best hyperparameters & evaluate on hold-out set.

2. Preprocessing Data

[DATACAMP]

Dealing with categorical features

- scikit learn will not accept categorical features by default
- Need to encode categorical features numerically.
- Convert to 'dummy variables'.
 - 0: Observation was NOT that category.
 - 1: Observation was of that category.

Dummy variables

Example

Origin		Origin-Asia	Origin-Europe	Origin-US
US	→	0	0	1
Europe		0	1	0
Asia		1	0	0

- You will be thinking that you have created dummy variables correctly but you are wrong: you have to drop one column because if we include all 3 columns that generate complexity and affect the model's performance.
- How, we will know that origin is of Europe if we have drop that column?

→ So,

origin	origin-US	Origin-Asia
US	1	0
Europe	0	0
Asia	0	1

The answer is if US & Asia column has 0 value than that data is of Europe column.

Dealing with Categorical features in ~~Pandas~~ Python

- Scikitlearn : OneHotEncoder()
- Pandas : get_dummies()

Encoding dummy variables

```
import pandas as pd  
df = pd.read.csv('auto.csv')  
df_origin = pd.get_dummies(df)  
  
df_origin = df_origin.drop('origin-Asia', axis=1)  
print(df_origin.head())
```

Linear Regression with dummy variables

```
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import Ridge
```

```
x_train, x_test, y_train, y_test = train_test_split  
(X, y, test_size=0.3, random_state=42)
```

```
ridge = Ridge(alpha=0.5, normalize=True),  
fit(x_train, y_train)
```

```
ridge.score(x_test, y_test)
```

↳ output : 0.719064519022

NOTE:-

- when you are creating dummy variables at that time also you can drop/remove unneeded columns.
- df_region = pd.get_dummies(data=df,
drop_first=True)

Handling Missing Data

→ PIMA Indians dataset

```
df = pd.read_csv('diabetes.csv')  
df.info()
```

→ using .info() method we didn't see any missing values.

→ So, let's see the data

```
print(df.head())
```

→ using above line of code we found that missing values are denoted as 0 here.

Dropping missing data

```
df.Pregnancies.replace(0, np.nan, inplace=True)
```

```
df.SkinThickness.replace(0, np.nan, inplace=True)
```

```
df.BloodPressure.replace(0, np.nan, inplace=True)
```

```
df.info()
```

```
df = df.dropna()
```

```
df.shape
```

Imputing missing data

- Making an educated guess about the missing values
- Example: Using the mean of the non-missing entries.

```
from sklearn.preprocessing import Imputer  
imp = Imputer(missing_values='NaN',  
               strategy='mean', axis=0)  
imp.fit(x)  
x = imp.transform(x)
```

Imputing within a pipeline

```
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import Imputer  
imp = Imputer(missing_values='NaN',  
               strategy='mean', axis=0)
```

```
logreg = LogisticRegression()  
steps = [('imputation', imp),  
         ('Logistic_regression'), logreg]
```

```
pipeline = Pipeline(steps)
```

```
X_train, X_test, y_train, y_test = train_test_split  
(X, y, test_size=0.3, random_state=42)
```

`pipeline.fit(X_train, y_train)`

`y_pred = pipeline.predict(X_test)`

`pipeline.score(X_test, y_test)`

↳ output : 0.7532467532 ...

• Centreing and Scaling

why scale your data?

- many models use some form of distance to inform them.
- Features on larger scales can unduly influence the model.
- Example: K-NN uses distance explicitly when making predictions.
- we want features to be on a similar scale.
- Normalizing (or scaling & centreing)

ways to normalize your data?

- Standardization: Subtract the mean & divide by variance.
All features are centred around zero and have variance one.
- Can also subtract the minimum & divide by the range.
- Minimum zero & maximum one.

- Can also normalize so the data ranges from -1 to +1.

Scaling in scikit-learn

```
from sklearn.preprocessing import scale
X_scaled = scale(X)
```

np.mean(x), np.std(x)

↳ output: (8.1432..., 15.7265)

np.mean(X_scaled), np.std(X_scaled)

↳ output: (2.5466e-15, 1.0)

Scaling in a pipeline

```
from sklearn.preprocessing import StandardScaler
steps = [('scaler', StandardScaler()), ('knn', KNeighborsClassifier())]
```

pipeline = Pipeline(steps)

X_train, X_test, y_train, y_test = train_test_split

(X, y, test_size=0.2, random_state=0)

knn_scaled = pipeline.fit(X_train, y_train)

y_pred = pipeline.predict(X_test)

accuracy_score(y_test, y_pred)

↳ output: 0.956

knn_unscaled = KNeighborsClassifier().fit(X_train,y_train)
knn_unscaled.score(X_test, y-test)
↳ output: 0.92

* As a result, scaling improves the accuracy.

CV and scaling in a pipeline

steps = [('scaler', StandardScaler()),
 ('knn', KNeighborsClassifier())]

pipeline = Pipeline(steps)

parameters = {'knn__n_neighbors': np.arange(1,50)}

X_train, X_test, y_train, y-test = train_test_split
(X, y, test_size=0.2, random_state=21)

cv = GridSearchCV(pipeline, param_grid=parameters)

cv.fit(X_train, y_train)

y-pred = cv.predict(X-test)

print(cv.best_params_)

↳ output: {'knn__n_neighbors': 41}

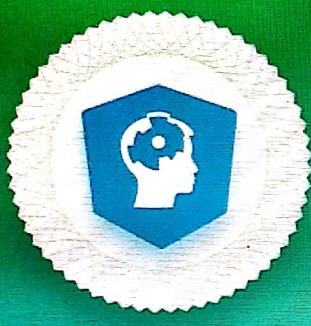
print(cv.score(X-test, y-test))

↳ output: 0.956

print(classification_report(y-test, y-pred))

↳ output: precision recall f1-score support

	0	1		
0	0.97	0.90	0.93	39
1	0.95	0.99	0.97	75
avg / total	0.96	0.96	0.96	114



STATEMENT OF ACCOMPLISHMENT

#6000892

HAS BEEN AWARDED TO

Junaid Khan

FOR SUCCESSFULLY COMPLETING

Supervised Learning with scikit-learn

COMPLETED ON

Jun 10, 2020



DataCamp