ML-C4

22.07.14

Feature_Engineering

Feature ->

1. Continuous Feature : 2차원 실수 배열로 각열이 데이터포

인트를 설명하는 연속형특성

2. 범주형 특성 & 이산형 특성
: 보통 숫자 값이 아니다.
-> 범주형 특성 : 상품 브랜드, 색상, 판매 분류

특정 어플리케이션에 가장 적합한 데이터 표현을 찾는 것을 특성공학 (Feature_Engineering)이라하며, 당명하는 주요 작업 중 하나.

Categorical_Value

1994년 인구조사 데이터베이스에서 추출한 성인 소득데이터 셋

-> 어떤 근로자의 수입이 5만달러를 초과하는지, 그 이하 일지를 예측하는 것.

근로자 나이, 고용형태, 교육수준, 성별, 주당 근로시간, 직업등의 특성이 존재한다.

One-Hot-Encoding(categorical -> continuous)

```
import pandas as pd
import os
data = pd.read_csv(os.path.join(mglearn.datasets.DATA_PATH,
"adult.data"),header=None,index_col=False,
             names=['age','workclass','fnlwgt','education','education-num','marital-
status','occupation','relationship','race','gender',
                 'captial-gain','caplital-loss','hours-per-week','native-country','income'])
data = data[['age','workclass','education','gender','hours-per-
week','occupation','income']]

display(data.head())
```

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K

```
data.gender.value_counts()
```

```
Male 21790 Female 10771 Name: gender, dtype: int64
```

One-Hot-Encoding(categorical -> continuous)

```
import pandas as pd
import os
data = pd.read_csv(os.path.join(mglearn.datasets.DATA_PATH, "adult.data"),header=None,index_col=False,
                names=['age','workclass','fnlwgt','education','education-num','marital-status','occupation','relationship','race','gender',
                    'captial-gain','caplital-loss','hours-per-week','native-country','income'])
data = data[['age','workclass','education','gender','hours-per-week','occupation','income']]
data_dum = pd.get_dummies(data)
print(data_dum.columns)
```

```
Index(['age', 'hours-per-week', 'workclass_ ?',
'workclass_ Federal-gov', 'workclass_ Local-gov',
'workclass_ Never-worked', 'workclass_ Private',
'workclass_ Self-emp-inc', 'workclass_ Self-emp-not-inc', 'workclass_ State-gov', 'workclass_ Without-pay', 'education_ 10th', 'education_ 11th',
'education_ 12th', 'education_ 1st-4th', 'education_ 5th-6th', 'education_ 7th-8th', 'education_ 9th',
'education_ Assoc-acdm', 'education_ Assoc-voc',
'education_ Bachelors', 'education_ Doctorate',
'education_ HS-grad', 'education_ Masters',
'education_ Preschool', 'education_ Prof-school',
'education_ Some-college', 'gender_ Female', 'gender_ Male', 'occupation_ ?', 'occupation_ Adm-clerical',
'occupation_ Armed-Forces', 'occupation_ Craft-repair', 'occupation_ Exec-managerial', 'occupation_ Farming-fishing', 'occupation_ Handlers-cleaners',
'occupation_ Machine-op-inspct', 'occupation_ Other-service', 'occupation_ Priv-house-serv', 'occupation_ Prof-specialty', 'occupation_ Protective-serv',
'occupation_ Sales', 'occupation_ Tech-support',
'occupation_ Transport-moving', 'income_ <=50K',
'income_ >50K'], dtype='object')
```

```
display(data_dum.head())
#5개 항목까지 표시
```

Pandas에서는 열 슬라이싱 할 시 마지막 범위를 포함한다!

One-Hot-Encoding(categorical -> continuous)

```
data_dum = pd.get_dummies(data)

features = data_dum.loc[:,'age':'occupation_ Transport-moving']
#dataframe에서 Numpy 배열로 추출하는데, age열 부터 transport-moving까지
뽑아낸다.
X= features.values #Numpy로 변환하는 코드는 dataframe.values 메소드이다.
y=data_dum['income_ >50K'].values

print(X.shape)
print(y.shape)
```


features = data_dum.loc[:,'age':'occupation_ Transport-moving']
#dataframe에서 Numpy 배열로 추출하는데, age열 부터 transport-moving까지
뽑아낸다.
X= features.values #Numpy로 변환하는 코드는 dataframe.values 메소드이다.
y=data_dum['income_ >50K'].values


```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
print("{:.3f}".format(logreg.score(X_test,y_test)))
```

```
(32561, 44) (32561,)
```

```
0.809
```

훈련데이터와 테스트데이터의 데이터프레임을 동일하게 가지고 가고, 분리하는게 속성 중복이나 속성개수가 달라지는 오류에 대비할 수 있다.

One-Hot-Encoding(categorical -> continuous)

```
df=pd.DataFrame({'num_f' :
[0,1,2,1],'cate_f' :
['socks','fox','socks','box']})

display(df)
```

	num_f	cate_f
0	0	socks
1	1	fox
2	2	socks
3	1	box

```
data_dum = pd.get_dummies(data)

df=pd.DataFrame({'num_f' : [0,1,2,1],'cate_f' :
['socks','fox','socks','box']})

df_dum=pd.get_dummies(df)
display(df_dum)
```

	num_f	cate_f_box	cate_f_fox	cate_f_socks
0	0	0	0	1
1	1	0	1	0
2	2	0	0	1
3	1	1	0	0

Pandas의 get_dummies 메소드는 데이터가 문자열 일 경우에만 가변수로 만들어주고, 숫자 일시에는 그대로 유지한다.

One-Hot-Encoding(categorical -> continuous)

숫자 데이터이지만, 범주형분류의 데이터 일경우 그 열을 문자열로 astype메소드를 통해 변환 시킨 후 getdummies의 매개변수에서 Coulumns=[]를 지정해줌으로써, 가변수로 만들수있다.

```
df=pd.DataFrame({'num_f' : [0,1,2,1],'cate_f' :
['socks','fox','socks','box']})

df['num_f']=df['num_f'].astype(str)
df_dum=pd.get_dummies(df,columns=['num_f','cate_f'
])
display(df_dum)
```

	num_f_0	num_f_1	num_f_2	cate_f_box	cate_f_fox	cate_f_socks
0	1	0	0	0	0	1
1	0	1	0	0	1	0
2	0	0	1	0	0	1
3	0	1	0	1	0	0

One-Hot-Encoding(column_transformer)

```
from sklearn.preprocessing import MinMaxScaler, QuantileTransformer, StandardScaler, PowerTransformer
From sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("Scaling",StandardScaler(),['age','hours-per-week']),("onehot",OneHotEncoder(sparse=False),['workclass','education','gender','occupation'])])

#("description",사용할 스케일링 메소드,['적용할 열들']),(똑같이 한다.)

data_features = data.drop("income",axis=1)

X_train , X_test, y_train, y_test = train_test_split(data_features,data.income,random_state=0)

ct.fit(X_train)
X_train_trans=ct.transform(X_train)

X_train_trans.shape
```

```
(24420, 44)
```

```
ct.fit(X_train)
X_train_trans=ct.transform(X_train)

logred = LogisticRegression()
logred.fit(X_train_trans,y_train)
X_test_trans = ct.transform(X_test)

logred.score(X_test_trans,y_test)
```

```
0.8088686893502027
```

연속형 값들에 StandardScaler로 스케일링 처리를 하였고, 범주형 열들에는 OneHotEncoder로 변환해주었다.

연속형 값에 scaling을 해주어도 이 데이터셋에서 크나큰 성능 차이는 없다.

One-Hot-Encoding(make_column_transformer)

```
ct = ColumnTransformer([("Scaling",StandardScaler(),['age','hours-per-week']),("onehot",OneHotEncoder(sparse=False),['workclass','education','gender','occupation'])])
```

```
ct = make_column_transformer((['age','hours-per-week'],StandardScaler()),(['workclass','education','gender','occupation'],OneHotEncoder(sparse=False)))
```

클래스이름을 지정해주지 않아도, 자동으로 각 단계에 이름을 붙여준다.

구간분할

```
from sklearn.preprocessing import KBinsDiscretizer
kb = KBinsDiscretizer(n_bins=10,strategy='uniform')
kb.fit(X)
print(kb.bin_edges_)
```

```
[array([-2.9668673 , -2.37804841, -1.78922951, -
1.20041062, -0.61159173, -0.02277284, 0.56604605,
1.15486494, 1.74368384, 2.33250273, 2.92132162])]
```

1개의 특성을 10개의 특성으로 분할 한다.

```
X_bin = kb.transform(X)
X_bin
```

```
<120x10 sparse matrix of type '<class
'numpy.float64'>' with 120 stored elements in
Compressed Sparse Row format>
```

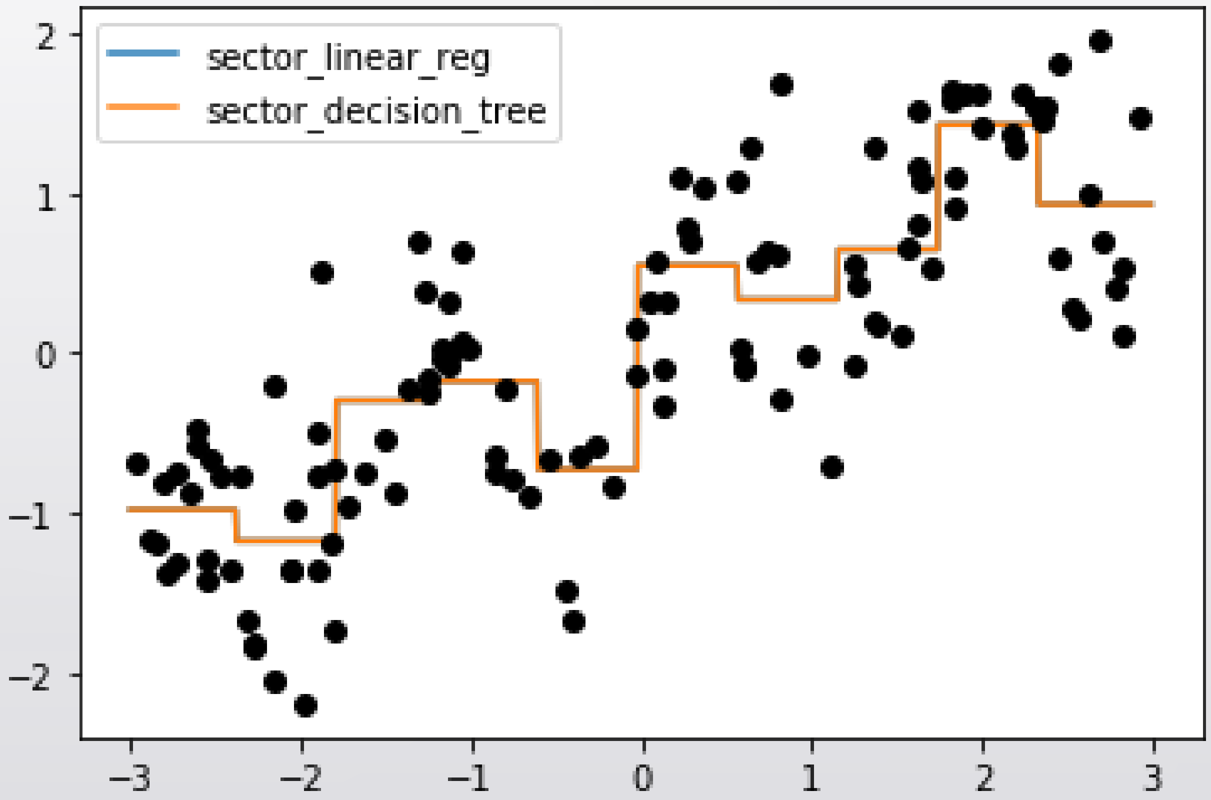
구간분할(Continous -> Categorical)

```
print(X[:10])
print(X_bin.toarray()[:10])
```

```
[[-0.75275929] [ 2.70428584] [ 1.39196365] [ 0.59195091] [-2.06388816]
[-2.06403288] [-2.65149833] [ 2.19705687] [ 0.60669007] [ 1.24843547]]
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
```

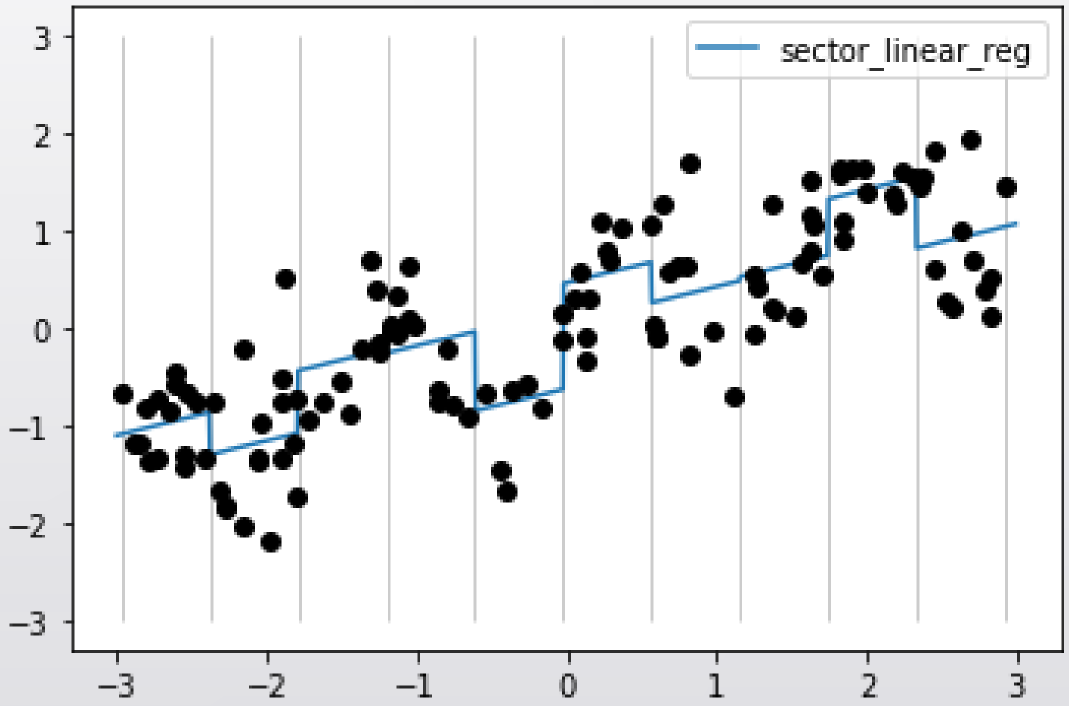
구간분할(10개구간, linear_regression vs decision_tree)

```
X, y = mglearn.datasets.make_wave(n_samples=120)
line = np.linspace(-3,3,1000,endpoint=False).reshape(-1,1)
kb = KBinsDiscretizer(n_bins=10,strategy='uniform',encode='onehot-dense')
kb.fit(X)
X_bin = kb.transform(X)

line_bin = kb.transform(line)

reg = LinearRegression().fit(X_bin,y)
plt.plot(line,reg.predict(line_bin),label='sector_linear_reg')
reg = DecisionTreeRegressor(min_samples_split=3).fit(X_bin,y)
plt.plot(line,reg.predict(line_bin),label='sector_decision_tree')
plt.plot(X[:,0],y,'o',c='k')
plt.legend()
```



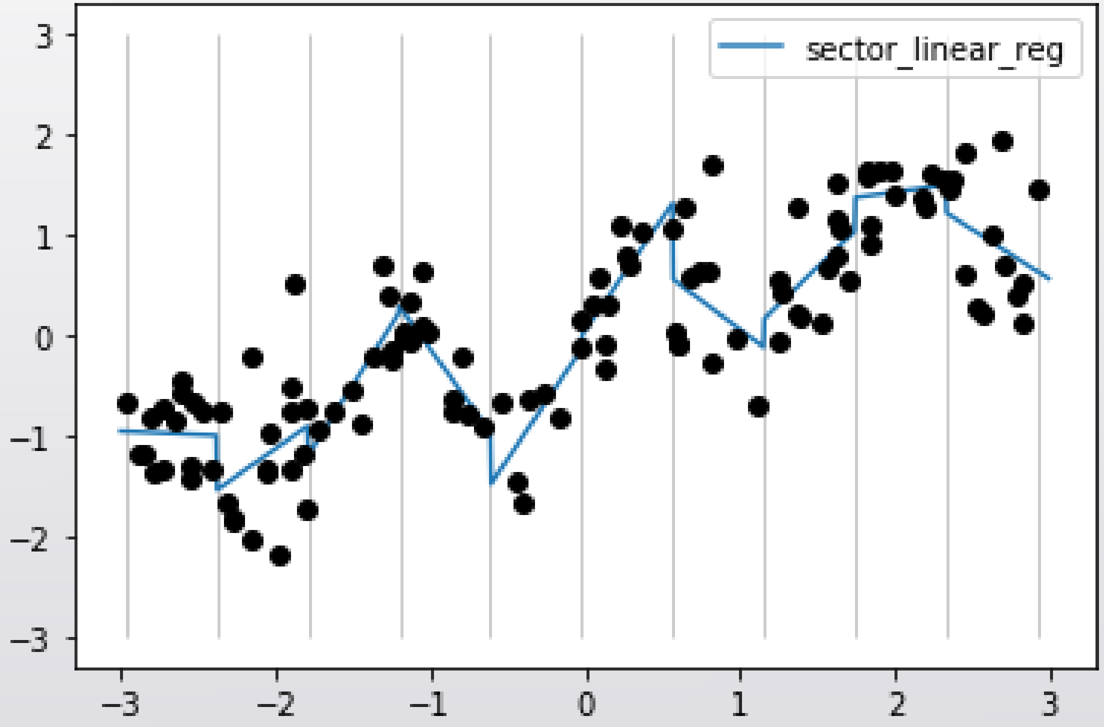
구간분할(상호작용과 다항식)

```
kb = KBinsDiscretizer(n_bins=10,strategy='uniform',encode='onehot-dense')
kb.fit(X)
X_bin = kb.transform(X)
X_bin_trans = np.hstack([X_bin,X])


line_bin = kb.transform(line)
line_bin_trans = np.hstack([line_bin,line])

reg = LinearRegression().fit(X_bin_trans,y)
plt.plot(line,reg.predict(line_bin_trans),label='sector_linear_reg')
plt.vlines(kb.bin_edges_[0],-3,3,linewidth=1,alpha=.2)
plt.plot(X[:,0],y,'o',c='k')
plt.legend()
```



구간분할(상호작용과 다항식)

```
kb = KBinsDiscretizer(n_bins=10,strategy='uniform',encode='onehot-dense')
kb.fit(X)
X_bin = kb.transform(X)
X_bin_trans = np.hstack([X_bin,X*bin])


line_bin = kb.transform(line)
line_bin_trans = np.hstack([line_bin,line*line_bin])

reg = LinearRegression().fit(X_bin_trans,y)
plt.plot(line,reg.predict(line_bin_trans),label='sector_linear_reg')
plt.vlines(kb.bin_edges_[0],-3,3,linewidth=1,alpha=.2)
plt.plot(X[:,0],y,'o',c='k')
plt.legend()
```



구간분할(상호작용과 다항식)

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=10,include_bias=False)
poly.fit(X)
X_poly= poly.transform(X)

X_poly.shape
```

```
(120, 10)
```

```
print(X[:5])
print(X_poly[:5])
```

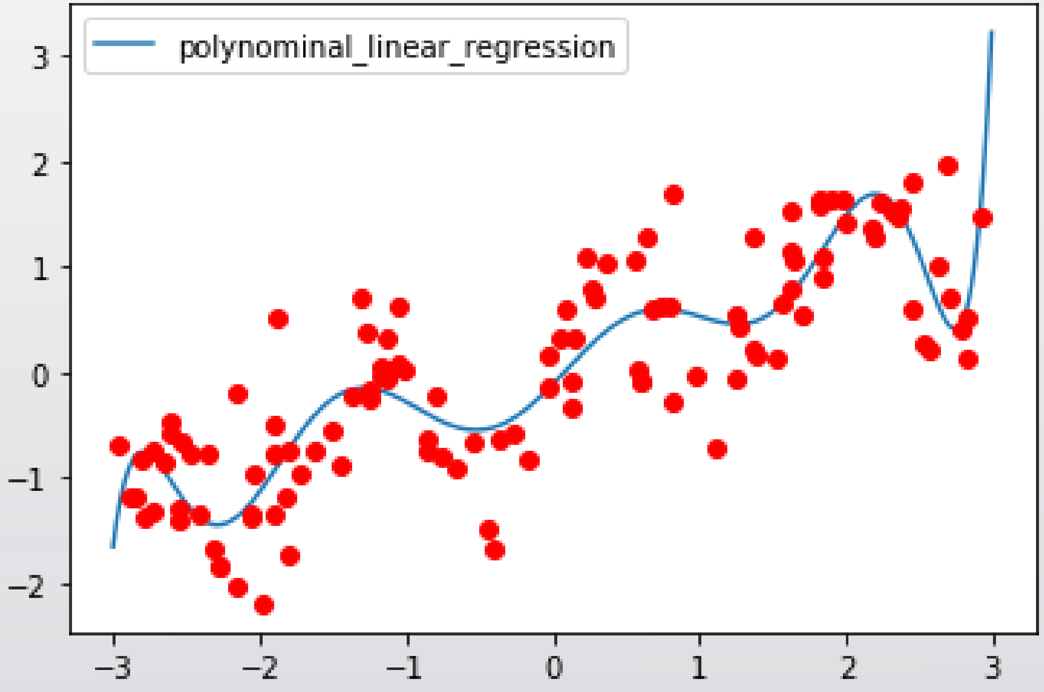
```
[[-0.75275929] [ 2.70428584] [
1.39196365] [ 0.59195091] [-
2.06388816]] [[-7.52759287e-01
5.66646544e-01 -4.26548448e-01
3.21088306e-01 -2.41702204e-01
1.81943579e-01 -1.36959719e-01
1.03097700e-01 -7.76077513e-02
5.84199555e-02] [ 2.70428584e+00
7.31316190e+00 1.97768801e+01
5.34823369e+01 1.44631526e+02
3.91124988e+02 1.05771377e+03
2.86036036e+03 7.73523202e+03
2.09182784e+04] [ 1.39196365e+00
1.93756281e+00 2.69701700e+00
3.75414962e+00 5.22563982e+00
7.27390068e+00 1.01250053e+01
1.40936394e+01 1.96178338e+01
2.73073115e+01] [ 5.91950905e-01
3.50405874e-01 2.07423074e-01
1.22784277e-01 7.26822637e-02
4.30243318e-02 2.54682921e-02
1.50759786e-02 8.92423917e-03
5.28271146e-03] [-2.06388816e+00
4.25963433e+00 -8.79140884e+00
1.81444846e+01 -3.74481869e+01
7.72888694e+01 -1.59515582e+02
3.29222321e+02 -6.79478050e+02
1.40236670e+03]]
```

```
print(poly.get_feature_names())
```

```
['x0', 'x0^2', 'x0^3', 'x0^4', 'x0^5', 'x0^6', 'x0^7', 'x0^8', 'x0^9', 'x0^10']
```

구간분할(상호작용과 다항식)

```
poly = PolynomialFeatures(degree=10,include_bias=False)
poly.fit(X)
X_poly= poly.transform(X)

reg = LinearRegression().fit(X_poly,y)
line_poly = poly.transform(line)
plt.plot(line,reg.predict(line_poly),label='polynominal_linear_regression')
plt.plot(X[:,0],y,'o',c='r')
plt.legend(loc='best')
plt.show()
```



소규모 데이터셋에 이런 고차원특성을 사용할 시, 굉장히
민감하게 동작한다.

구간분할(2개 combination)

```
boston = load_boston()
X_train, X_test, y_train, y_test =
train_test_split(boston.data,boston.target,random_state=0)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

poly = PolynomialFeatures(degree=2).fit(X_train_scaled)
X_train_poly = poly.transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

print ( X_train_scaled.shape , X_train_poly.shape)
```

```
(379, 13) (379, 105) #원본 특성들과 2개의 조합 + 절편까지 포함해서 105개이다.
```

구간분할(Ridge)

```
boston = load_boston()
X_train, X_test, y_train, y_test =
train_test_split(boston.data,boston.target,random_state=0)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

poly = PolynomialFeatures(degree=2).fit(X_train_scaled)
X_train_poly = poly.transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)

ridge = Ridge().fit(X_train_scaled,y_train)
print("ridge_poly_x : {:.2f}".format(ridge.score(X_test_scaled,y_test)))
ridge = Ridge().fit(X_train_poly,y_train)
print("ridge_poly_o : {:.2f}".format(ridge.score(X_test_poly,y_test)))
```

```
ridge_poly_x : 0.62 ridge_poly_o : 0.75
```

```
rf = RandomForestRegressor(n_estimators=100,random_state=0).fit(X_train_scaled,y_train)

print("randomforest_poly_x : {:.2f}".format(rf.score(X_test_scaled,y_test)))
rf = RandomForestRegressor(n_estimators=100,random_state=0).fit(X_train_poly,y_train)
print("randomforest_poly_o : {:.2f}".format(rf.score(X_test_poly,y_test)))
```

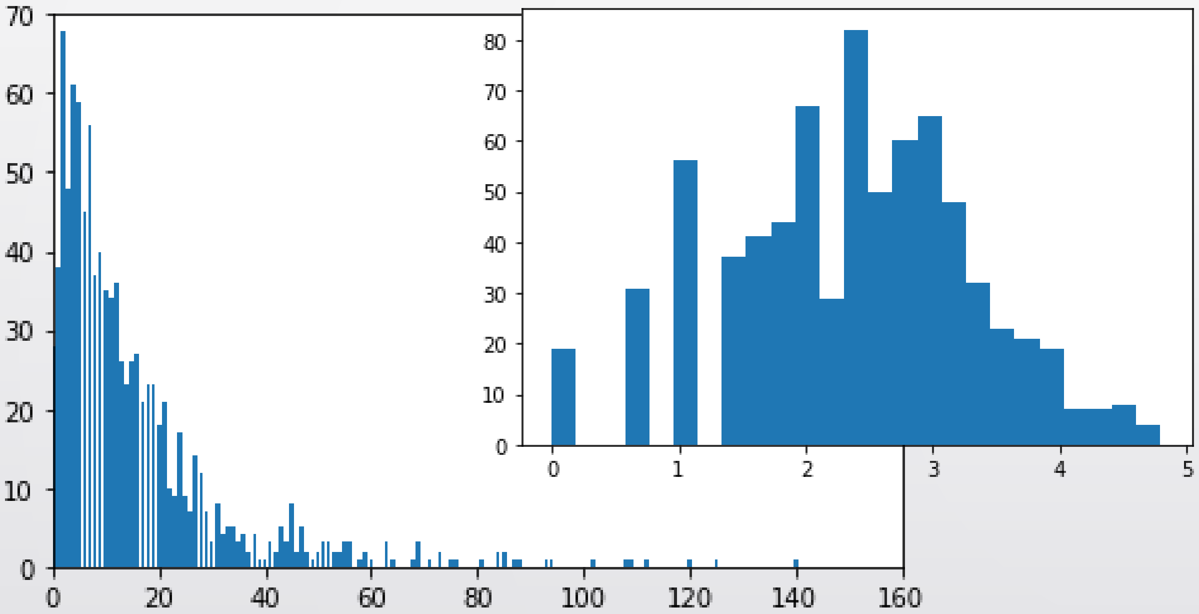
```
randomforest_poly_x : 0.80 randomforest_poly_o : 0.77
#트리모델은 특성을 추가하지 않아도 성능이 잘나온다.
```

일변량 비선형 변환

대부분의 모델은 각 특성이 정규분포와 비슷할 때 최고의 성능을 발휘한다. ->log, exp

```
rnd = np.random.RandomState(0) # seed
X_org = rnd.normal(size=(1000,3)) # 1000,3 짜리 정규분포 형태 난수 발생
w=rnd.normal(size=3)

X=rnd.poisson(10*np.exp(X_org)) # poisson은 데이터분포가 극단적으로 모인상태
y=np.dot(X_org, w)
print(np.bincount(X[:,0]))

plt.xlim(0,160)
plt.ylim(0,70)

bins = np.bincount(X[:,0])
plt.bar(range(len(bins)),bins)
plt.show()

rid = Ridge().fit(X_train,y_train)
rid.score(X_test,y_test)
```

```
0.6224186236310756
```




```
X_train = np.log(X_train+1)
X_test = np.log(X_test+1)

rid = Ridge().fit(X_train,y_train)
rid.score(X_test,y_test)
```

```
0.8749342372887815
```

일변량 비선형 변환

특성 자동선택 ->일변량 통계, 모델 기반선택, 반복적 선택

모두 지도학습 방법이므로 최적값을 찾으려면 타겟이필요.

일변량 통계는 개개의 특성과 타깃 사이에 중요한 통계적
관계가 있는지를 계산하는 방법.

-> 분류에서는 분산 분석이라고 한다.
일변량 : 각 특성이 독립적으로 평가된다.
=> 다른 특성과 깊게 연관된 특성은 선택되지 않는다.

Sklearn에서는 분류에서는 f_classif / 회귀에서는
f_regression을 보통 선택하여 테스트하고,

계산한 p값에 기초하여 특성을 제외하는 방식을 선택한다.

P값이 크다는 것은 그 특성은 target과의 연관성이 적다는
것을 의미하여, 그 특성을 제외하는 방식으로 진행.

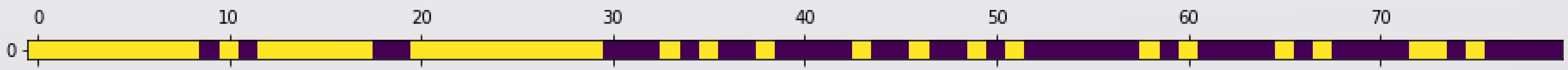
일변량 비선형 변환

```
from sklearn.feature_selection import f_classif, SelectPercentile
cancer = load_breast_cancer()

rng = np.random.RandomState(42)
noise = rng.normal(size=(len(cancer.data),50)) # 50개의 특성은 노이즈로 랜덤생성
X_w = np.hstack([cancer.data,noise]) # 80개중 30개는 원래 특성 + 50 노이즈

X_train, X_test, y_train, y_test = train_test_split(X_w,
cancer.target,random_state=0,test_size=.5)

select = SelectPercentile(score_func=f_classif,percentile=50)
select.fit(X_train,y_train)

X_train_select = select.transform(X_train)

print(X_train_select.shape)
```

```
(284, 40)
```


```
X_train, X_test, y_train, y_test = train_test_split(X_w,
cancer.target,random_state=0,test_size=.5)

select = SelectPercentile(score_func=f_classif,percentile=50)
select.fit(X_train,y_train)

X_train_select = select.transform(X_train)
```


```
mask = select.get_support()

print(mask.shape)

plt.matshow(mask.reshape(1,-1))

plt.yticks([0])
```




```
logi = LogisticRegression().fit(X_train_select,y_train)
print("selected_feature : {:.2f}".format(logi.score(X_test_select,y_test)))
logi.fit(X_train,y_train)
print("all_feature : {:.2f}".format(logi.score(X_test,y_test)))
```

```
selected_feature : 0.94 all_feature : 0.93
```

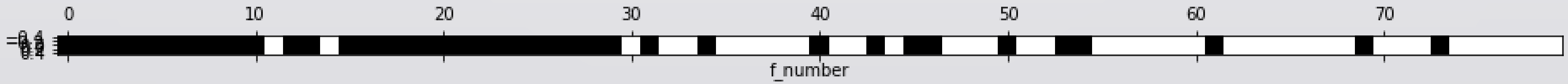
모델 기반 특성 선택

특성선택을 위한 지도학습 모델 1개 + 학습 및 테스트를 위한 모델 1개로 구성(같을 필요는 없다.)

```
from sklearn.feature_selection import SelectFromModel
select = 
SelectFromModel(RandomForestClassifier(n_estimators=100,random_state=42),t
hreshold="median")

select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
X_test_l1 = select.transform(X_test)
print("X_train.shape : ",X_train.shape)
print("X_train_l1.shape : ",X_train_l1.shape)

mask = select.get_support()
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("f_number")

lg = LogisticRegression().fit(X_train_l1,y_train)
lg.score(X_test_l1,y_test)
```

```
X_train.shape : (284, 80) X_train_l1.shape : (284, 40)
```

```
0.9508771929824561
```



반복적 특성 선택

1. 모든 특성을 가지고 있다가, 특성중요도가 낮은것을 하나씩 제거해나가면서, 종료조건 까지 반복 -> RFE

2. 특성을 가지고 있지 않다가, 하나씩 추가하면서 종료조건에 이를때까지 반복해나가는것.

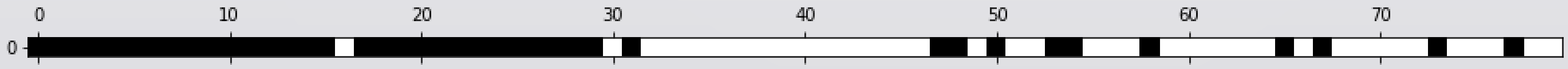
모델 기반 특성 선택과 같이 특성중요도를 제공해주는 모델이 필요하다.

```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=100,
random_state=42),n_features_to_select=40) #종료조건과 모델
select.fit(X_train,y_train)
mask=select.get_support()

plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.yticks([0])
```

```
X_train_ref=select.transform(X_train)
X_test_ref=select.transform(X_test)

lg=LogisticRegression().fit(X_train_ref, y_train)
lg.fit(X_train_ref, y_train)
lg.score(X_test_ref,y_test)
```

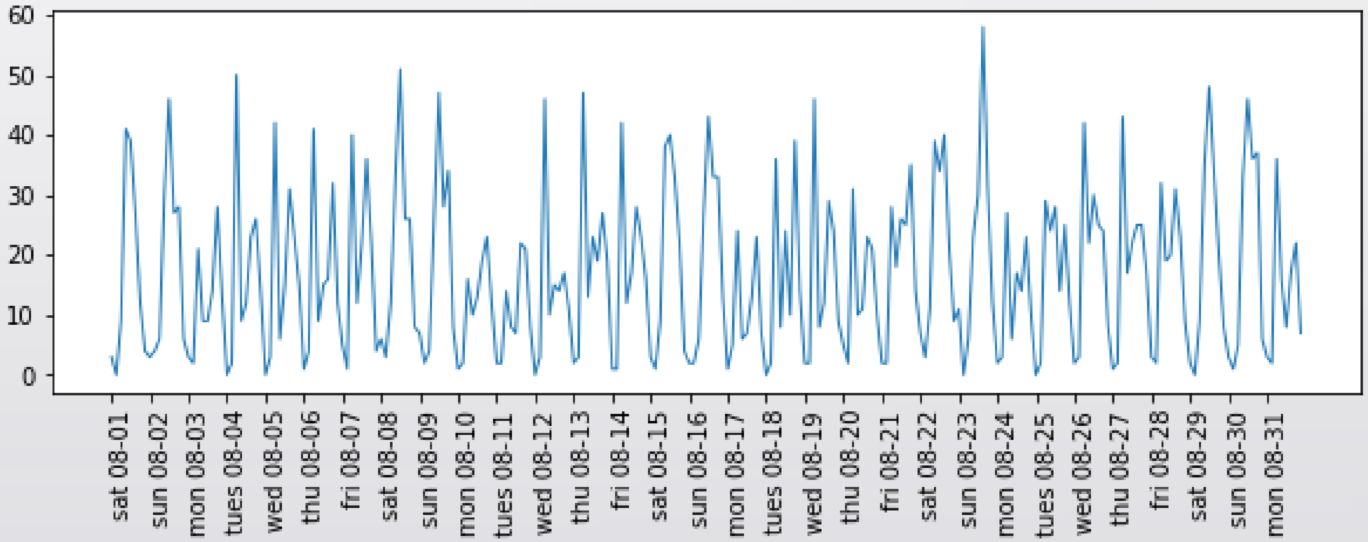
```
0.9508771929824561
```

RFE 방법은 80개의 특성에서 특성중요도가 낮은 특성을 하나씩 제거할때마다 재학습 시키므로 시간이 오래걸린다. (총 40번의 학습)



+@(전문가 지식 활용)

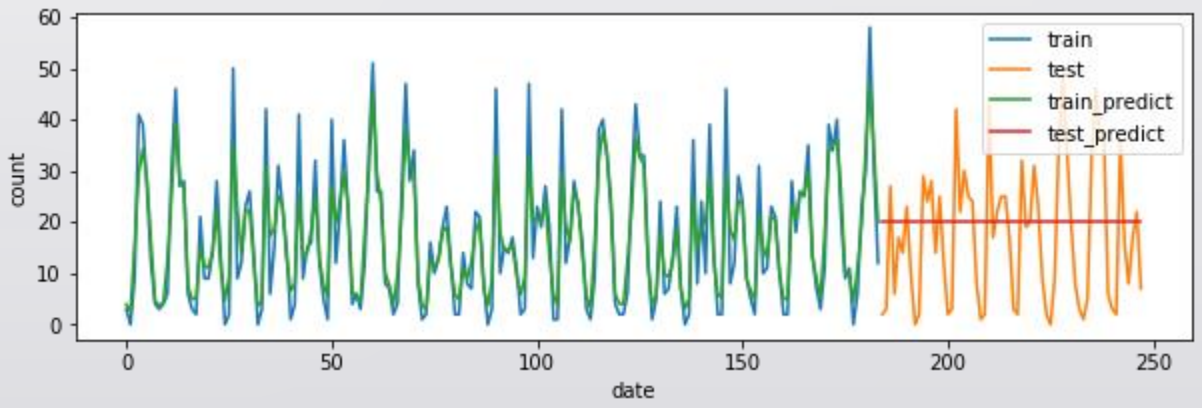
```
citibike = mglearn.datasets.load_citibike()

plt.figure(figsize=(10,3))
xticks = pd.date_range(start = citibike.index.min(),end=citibike.index.max(),freq='D ' ) #pandas의 datestamp로 읽어온다. 최소부터 최대
week=["sun","mon","tues","wed","thu","fri","sat"]
xticks_name = [week[int(w)]+d for w, d in zip(xticks.strftime("%w"),xticks.strftime(" %m-%d"))] #strftime은 datestamp를 str으로 변환시켜주며, 형식을 #
지정해줄 수 있다. %w는 0-6 까지 요일 / m은 달 d는 일

plt.xticks(xticks,xticks_name,rotation=90,ha="left")
plt.plot(citibike,linewidth=1)
```



+@(전문가 지식 활용) -> 날짜 전체를 학습(random_forest)

```
citibike = mglearn.datasets.load_citibike()

#타깃값 출력
y=citibike.values

X=citibike.index.astype("int64").values.reshape(-1,1) // 10**9 # 10^-9이므로 그만큼 나눠준다.

n_train = 184 # 3시간 당 측정이므로 24시간당 8개 즉, 23*8=184

def eval_on_features(features, target, regressor):
    X_train, X_test = features[:n_train],features[n_train:]
    y_train, y_test = target[:n_train],target[n_train:]
    regressor.fit(X_train, y_train)
    print("test_accuracy : {:.2f}".format(regressor.score(X_test,y_test)))
    y_pred = regressor.predict(X_test)
    y_pred_train = regressor.predict(X_train)

    plt.figure(figsize=(10,3))
    plt.plot(range(n_train),y_train,label="train")
    plt.plot(range(n_train,len(y_test)+n_train),y_test,label="test")
    plt.plot(range(n_train),y_pred_train,label="train_predict")
    plt.plot(range(n_train,len(y_test)+n_train),y_pred,label="test_predict")

    plt.legend(loc='best')
    plt.xlabel("date")
    plt.ylabel("count")


regressor = RandomForestRegressor(n_estimators=100,random_state=0)
eval_on_features(X,y,regressor)
```

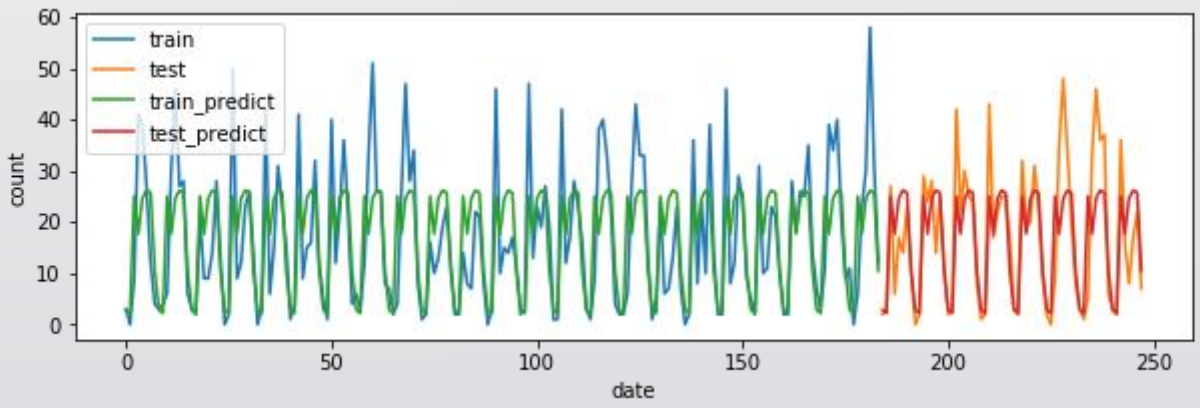
랜덤포레스트 모델은 훈련세트 특성 범위 밖으로 외삽할수 있는 능력이 없다. -> 테스트세트의 특성인 날짜값이 훈련세트의 특성인 날짜값 범위를 벗어나므로, 훈련세트의 마지막 타겟값만으로 예측을 할 수 밖에 없다.



+@(전문가 지식 활용)-> 시간을 학습

```
citibike = mglearn.datasets.load_citibike()

#타깃값 출력
y=citibike.values

X=citibike.index.astype("int64").values.reshape(-1,1) // 10**9 # 10^-9이므로 그만큼 나눠준다.

n_train = 184 # 3시간 당 측정이므로 24시간당 8개 즉, 23*8=184

def eval_on_features(features, target, regressor):
    X_train, X_test = features[:n_train],features[n_train:]
    y_train, y_test = target[:n_train],target[n_train:]
    regressor.fit(X_train, y_train)
    print("test_accuracy : {:.2f}".format(regressor.score(X_test,y_test)))
    y_pred = regressor.predict(X_test)
    y_pred_train = regressor.predict(X_train)

    plt.figure(figsize=(10,3))
    plt.plot(range(n_train),y_train,label="train")
    plt.plot(range(n_train,len(y_test)+n_train),y_test,label="test")
    plt.plot(range(n_train),y_pred_train,label="train_predict")
    plt.plot(range(n_train,len(y_test)+n_train),y_pred,label="test_predict")

    plt.legend(loc='best')
    plt.xlabel("date")
    plt.ylabel("count")


regressor = RandomForestRegressor(n_estimators=100,random_state=0)
X_hour = citibike.index.hour.values.reshape(-1,1)
eval_on_features(X_hour,y,regressor)
```

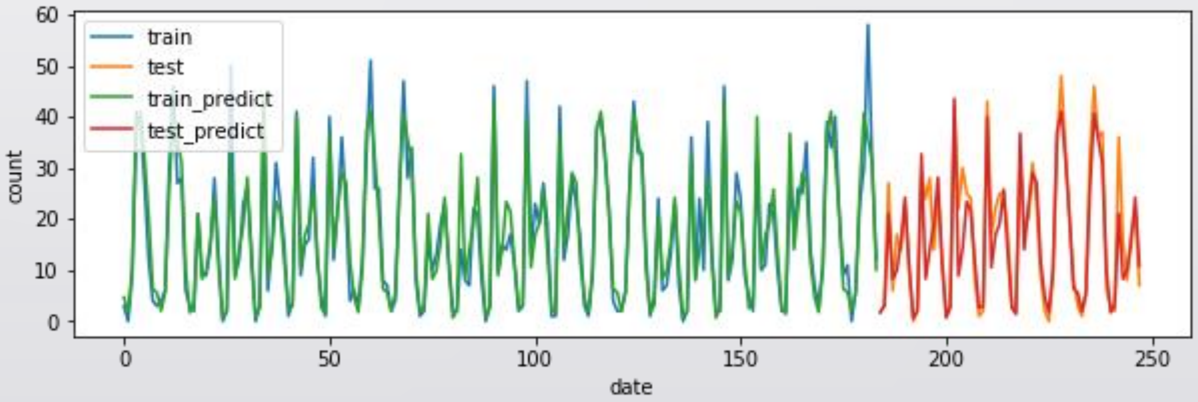
시간은 테스트세트나 학습세트 모두 24시간을 동일하나, 요일에 따른 패턴을 학습하지 못한다. 요일 개념을 추가해보자.



+@(전문가 지식 활용)-> 시간+ 요일

```
citibike = mglearn.datasets.load_citibike()

#타깃갑 출력
y=citibike.values

X=citibike.index.astype("int64").values.reshape(-1,1) // 10**9 # 10^-9이므로 그만큼 나눠준다.

n_train = 184 # 3시간 당 측정이므로 24시간당 8개 즉, 23*8=184

def eval_on_features(features, target, regressor):
    X_train, X_test = features[:n_train],features[n_train:]
    y_train, y_test = target[:n_train],target[n_train:]
    regressor.fit(X_train, y_train)
    print("test_accuracy : {:.2f}".format(regressor.score(X_test,y_test)))
    y_pred = regressor.predict(X_test)
    y_pred_train = regressor.predict(X_train)

    plt.figure(figsize=(10,3))
    plt.plot(range(n_train),y_train,label="train")
    plt.plot(range(n_train,len(y_test)+n_train),y_test,label="test")
    plt.plot(range(n_train),y_pred_train,label="train_predict")
    plt.plot(range(n_train,len(y_test)+n_train),y_pred,label="test_predict")

    plt.legend(loc='best')
    plt.xlabel("date")
    plt.ylabel("count")


regressor = RandomForestRegressor(n_estimators=100,random_state=0)
X_hour_week = np.hstack([citibike.index.dayofweek.values.reshape(-1,1)citibike.index.hour.values.reshape(-1,1)])
eval_on_features(X_hour_week,y,regressor)
```

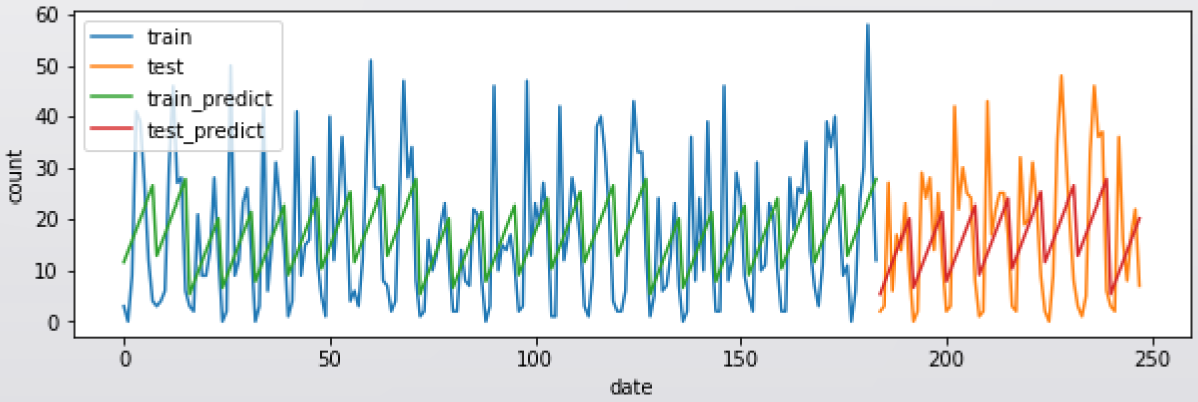
성능에 큰 상승이 있다.



+@(전문가 지식 활용)-> 시간+ 요일 -> LinearRegession

```
citibike = mglearn.datasets.load_citibike()

#타깃갑 출력
y=citibike.values

X=citibike.index.astype("int64").values.reshape(-1,1) // 10**9 # 10^-9이므로 그만큼 나눠준다.

n_train = 184 # 3시간 당 측정이므로 24시간당 8개 즉, 23*8=184

def eval_on_features(features, target, regressor):
    X_train, X_test = features[:n_train],features[n_train:]
    y_train, y_test = target[:n_train],target[n_train:]
    regressor.fit(X_train, y_train)
    print("test_accuracy : {:.2f}".format(regressor.score(X_test,y_test)))
    y_pred = regressor.predict(X_test)
    y_pred_train = regressor.predict(X_train)

    plt.figure(figsize=(10,3))
    plt.plot(range(n_train),y_train,label="train")
    plt.plot(range(n_train,len(y_test)+n_train),y_test,label="test")
    plt.plot(range(n_train),y_pred_train,label="train_predict")
    plt.plot(range(n_train,len(y_test)+n_train),y_pred,label="test_predict")

    plt.legend(loc='best')
    plt.xlabel("date")
    plt.ylabel("count")
```

시간과 요일은 범주형 변수이지만, 선형회귀는 연속형 변수로 취급하여 안좋은 성능을 가진다. 연속형-> 범주형으로 바꿔주는 OnehotEncoding



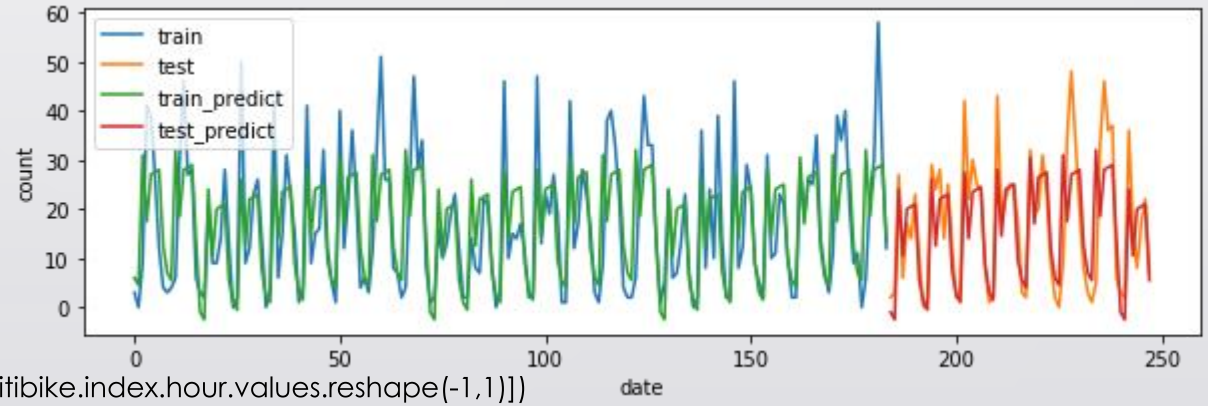

```
regressor =LinearRegression()
X_hour_week = np.hstack([citibike.index.dayofweek.values.reshape(-1,1)citibike.index.hour.values.reshape(-1,1)])
eval_on_features(X_hour_week,y,regressor)
```

+@(전문가 지식 활용)-> 시간+ 요일 -> LinearRegession OneHotEncoding

```
citibike = mglearn.datasets.load_citibike()

#타깃값 출력
y=citibike.values

X=citibike.index.astype("int64").values.reshape(-1,1) // 10**9 # 10^-9이므로 그만큼 나눠준다.

n_train = 184 # 3시간 당 측정이므로 24시간당 8개 즉, 23*8=184

def eval_on_features(features, target, regressor):
    X_train, X_test = features[:n_train],features[n_train:]
    y_train, y_test = target[:n_train],target[n_train:]
    regressor.fit(X_train, y_train)
    print("test_accuracy : {:.2f}".format(regressor.score(X_test,y_test)))
    y_pred = regressor.predict(X_test)
    y_pred_train = regressor.predict(X_train)

    plt.figure(figsize=(10,3))
    plt.plot(range(n_train),y_train,label="train")
    plt.plot(range(n_train,len(y_test)+n_train),y_test,label="test")
    plt.plot(range(n_train),y_pred_train,label="train_predict")
    plt.plot(range(n_train,len(y_test)+n_train),y_pred,label="test_predict")

    plt.legend(loc='best')
    plt.xlabel("date")
    plt.ylabel("count")

enc = OneHotEncoder(sparse=False)
X_hour_week =np.hstack([citibike.index.dayofweek.values.reshape(-1,1), citibike.index.hour.values.reshape(-1,1)])
X_hour_week_one=enc.fit_transform(X_hour_week)
print(enc.get_feature_names().shape)

regressor = LinearRegression()
eval_on_features(X_hour_week_one,y,regressor)
```

```
(15,) test_accuracy : 0.61
```

7(0-6:요일) + 8(24/3)



+@(전문가 지식 활용)-> 시간+ 요일 -> LinearRegession OneHotEncoding + 상호작용 (조합)

```
enc = OneHotEncoder(sparse=False)
X_hour_week =np.hstack([citibike.index.dayofweek.values.reshape(-1,1),
citibike.index.hour.values.reshape(-1,1)])
X_hour_week_one=enc.fit_transform(X_hour_week)
poly_t = PolynomialFeatures(degree=2,interaction_only=True,include_bias=False)
X_hour_week_one_poly = poly_t.fit_transform(X_hour_week_one)

print(len(poly.get_feature_names()))

regressor = Ridge()
eval_on_features(X_hour_week_one_poly,y,regressor)
```

```
105 test_accuracy : 0.85
```