Scikit-Learn Document

1. Preprocessing

1.1 Train, Test datsets split

We us train_test_split for preventing overfitting problem.
Source : train_test_split method

```
from sklearn.model_selection import StratifiedKFold, train_test_split

X_train, X_holdout, y_train, y_holdout = train_test_split(df_train, y, train_size = 0.7, test_size = 0.3, rand
om_state = 17)
```

1.2 get dummies

We use get_dummies to convert categorical variable into dummy/indicator variables.

```
df = pd.get_dummies(df, columns = categoricals)
```

1.3 neHotEncoder

Same method working same as get_dummies()

```
Encoder = OneHotEncoder()

Encoder.fit_transform(X_train)
Encoder.fit(X_valid)

cat_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown = 'ignore')),
])
```

1.4 SimpleImputer

변수에 특정한 방법을 적용하여 NaN값을 일괄 처리한다.

```
from sklearn.impute import SimpleImputer

Imputer = SimpleImputer(strategy = 'most_frequent/median/mean')

Imputer.fit_transform(X_train) # fit + transform, fit과 transform을 분리해서 사용 가능하다.
Imputer.fit(X_valid)

cat_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown = 'ignore')),
])
```

1.5 polynomial_fit

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X)
```

1.6 Scalers

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler
X_train_scaled = scaler().fit_transform(X_train)
X_holdout_scaled = scaler().fit_trainsform(X_holdout)
```

2. Feature Engineering

2.1 Mutual information(Discrete features)

Discrete features의 상관관계를 파악한다. Continuous features의 Correlation metrics와 동일한 의미를 가지며, 이 metric에서 좋은 점수를 얻지 못한 features는 제외하거나 다른 feature와 상관관계를 확인하여 새로운 변수량을 생성할 필요가 있음.

```
from sklearn.feature_selection import mutual_info_regression

def make_mi_scores(X, y, discrete_features):
    mi_scores = mutual_info_regression(X, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("Mutual Information Scores")
```

2.2 Creating Features

새로운 변수를 만드는 행위는 매우 중요하다 예를들어 정규화를 이루고 있지 않는 변수를 정규화를 시킨다던가, 더 좋은 상관관계를 가지고 있는 변수를 생성하여 모델에 추가하는 행위는 더욱 생산성을 높임

1. Make new columns : df[col3] = df[col1]/df[col2]
2. Do powers and logarithms to make datasets gets normalize : df[[gcol] = df[col].apply(np.log)
3. Count : Target variable과 상관관계가 높지 않는 변수들을 Count로 묶어 총 몇개가 있는지 표시하거나 Id별로 몇개의 0.0값이 있는지 확인하는 작업, sum() or gt(0).sum()을 사용
4. Str 연산자 : Str을 통해 Object type의 Insigth meaning을 추출한다. 예를들어 휴대전화에서 지역을, 혹은 등급+단계에서 각각의 feature로 분리하는 법
5. Group transform : Grouping 연산을 한 뒤에 각각을 mapping하여 datasets에 붙이기 보다 tranform을 사용하여 one line code로 비교 dataset feature를 생성한다.

2.3 Cluter Label as a Feature

특정 군집화를 하여 그룹을 지정해줌으로써 복잡한 관계를 한 차원 낮게 풀어줄 수 있다. K-means clustering을 적용하며 자연스러운 pandas bin() function으로 category variable을 생성한다.

3. Model Constructing

3.1 Pipe line

source : https://lsjsj92.tistory.com/579

```
from sklearn.pipeline import Pipeline

knn_pipe = Pipeline(
    [("scaler", StandardScaler()), ("knn", KNeighborsClassifier(n_jobs=-1))]
)

knn_params = {"knn__n_neighbors": range(1, 10)}

knn_grid = GridSearchCV(knn_pipe, knn_params, cv=5, n_jobs=-1, verbose=True)

knn_grid.fit(X_train, y_train)

knn_grid.best_params_, knn_grid.best_score_
```

3.2 ColumnTransformer

복수의 변수들을 적용할 Preprocessor마다 분산 적용하기 위해 사용.

```
# Preprocessing for numerical data

num_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'median')),
    ('scale', StandardScaler())
])

# Preprocessing for categorical data

cat_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown = 'ignore')),
])

# Preprocessing for discrete data

dis_transformer = Pipeline(steps = [
    ('imputer', SimpleImputer(strategy = 'most_frequent')),
    ('scale', StandardScaler())
])

# Bundle Preprocessing for all varaibles

preprocessor = ColumnTransformer(transformers = [
    ('num', num_transformer, num_cols),
    ('cat', cat_transformer, cat_cols),
    ('dis', dis_transformer, dis_cols)
])
```

3.3 Grid Search

Grid search is exploratory way to find hyper parameters making best score of model.

결론 : 그리드 서치를 하는 이유는 " 가장 우수한 성능을 보이는 모델의 하이퍼 파라미터를 찾기 위해서 ". 이유는 단순하다. 모든 경우의 수를 때려 넣어보고 가장 성능이 좋게 만드는 모델의 하이퍼 파라미터를 찾는거다.

Source : GridSearchCV

```
forest_params = {"max_depth": range(6, 12), "max_features": range(4, 19)}

forest_grid = GridSearchCV(forest, forest_params, cv=5, n_jobs=-1, verbose=True)

forest_grid.fit(X_train, y_train)

forest_grid.best_params_, forest_grid.best_score_  # ({'max_depth': 9, 'max_features': 6}, 0.951)

best_accuracy_model = forest_grid.best_estimator_
```

4. Model library

In [6]:
```
# Data Preprocessing

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler

# Model Construction

from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# Model

# 1) Linear Prediction

# 2) Classification



# Model Evaluation

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix

from sklearn.metrics import r2_score, mean_squaared_error, mean_absolute_error

from sklearn.model_selection import cross_val_score
```