algorithm-chains-and-pipelines-exercise

June 4, 2019

0.1 Pipeline

```
In [ ]: from sklearn.svm import SVC
        from sklearn.datasets import load_breast_cancer
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import MinMaxScaler

        #
        cancer = load_breast_cancer()
        X_train, X_test, y_train, y_test = train_test_split(
            cancer.data, cancer.target, random_state=0)

        #   ,
        scaler = MinMaxScaler().fit(X_train)
```

```
In [ ]: #
        X_train_scaled = scaler.transform(X_train)

        svm = SVC()
        #    SVM
        svm.fit(X_train_scaled, y_train)
        #
        X_test_scaled = scaler.transform(X_test)
        print(" : {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

0.1.1 Parameter

```
In [ ]: from sklearn.model_selection import GridSearchCV
        #     .   .
        param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
                      'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
        grid = GridSearchCV(SVC(), param_grid=param_grid, cv=5)
        grid.fit(X_train_scaled, y_train)
        print("   : {:.2f}".format(grid.best_score_))
        print(" : {:.2f}".format(grid.score(X_test_scaled, y_test)))
        print(" : ", grid.best_params_)
```

```
In [ ]: svm = SVC(C=1, gamma=1)
        #    SVM
        svm.fit(X_train_scaled, y_train)
        #
        X_test_scaled = scaler.transform(X_test)
        print(" : {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

```
In [ ]: mglearn.plots.plot_improper_processing()
```

```
In [ ]: from sklearn.pipeline import Pipeline
        pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC())])
```

```
In [ ]: pipe.fit(X_train, y_train)
```

```
In [ ]: print(" : {:.2f}".format(pipe.score(X_test, y_test)))
```

0.1.2 GridSearch with Pipeline

```
In [ ]: param_grid = {'svm__C': [0.001, 0.01, 0.1, 1, 10, 100],
                      'svm__gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

```
In [ ]: grid = GridSearchCV(pipe, param_grid=param_grid, cv=5) # 5*6*6 = 180
        grid.fit(X_train, y_train)
        print("   : {:.2f}".format(grid.best_score_))
        print("  : {:.2f}".format(grid.score(X_test, y_test)))
        print(" : {}".format(grid.best_params_))
```

```
In [ ]: mglearn.plots.plot_proper_processing()
```

```
In [ ]: import numpy as np
        rnd = np.random.RandomState(seed=0)
        X = rnd.normal(size=(100, 10000))
        y = rnd.normal(size=(100,))
```

```
In [ ]: from sklearn.feature_selection import SelectPercentile, f_regression

        select = SelectPercentile(score_func=f_regression, percentile=5).fit(X, y)
        X_selected = select.transform(X)
        print("X_selected.shape: {}".format(X_selected.shape))
```

```
In [ ]: from sklearn.model_selection import cross_val_score
        from sklearn.linear_model import Ridge
        print("   (): {:.2f}".format(
              np.mean(cross_val_score(Ridge(), X_selected, y, cv=5))))
```

```
In [ ]: def fit(self, X, y):
            X_transformed = X
            for name, estimator in self.steps[:-1]:
                print(name, estimator)
                #    fit transform
                X_transformed = estimator.fit_transform(X_transformed, y)
            #   fit
            self.steps[-1][1].fit(X_transformed, y)
            return self
```

```
In [ ]: fit(pipe, X, y)
```

```
In [ ]: def predict(self, X):
            X_transformed = X
            for step in self.steps[:-1]:
                #    transform
                X_transformed = step[1].transform(X_transformed)
            #   predict
            return self.steps[-1][1].predict(X_transformed)
```

make_pipleline

```
In [ ]: from sklearn.preprocessing import MinMaxScaler
        from sklearn.svm import SVC
        from sklearn.pipeline import make_pipeline
        #
        pipe_long = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC(C=100))])
        #
        pipe_short = make_pipeline(MinMaxScaler(), SVC(C=100))
```

```
In [ ]: print(" :\n{}".format(pipe_short.steps))
```

```
In [ ]: from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA

        pipe = make_pipeline(StandardScaler(), PCA(n_components=2), StandardScaler())
        print(" :\n{}".format(pipe.steps))
```

```
In [ ]:
```

```
In [ ]: # quiz - standardscaler, pca, logisticregression
        from sklearn.datasets import load_breast_cancer
        from sklearn.model_selection import train_test_split
        cancer = load_breast_cancer()
        X_train, X_test, y_train, y_test = train_test_split(cancer.data, \
                                cancer.target, test_size=0.3, random_state=77)
```

```
In [ ]: from sklearn.decomposition import PCA
        from sklearn.linear_model import LogisticRegression

        pipe = make_pipeline(StandardScaler(), PCA(n_components=2),
                LogisticRegression(solver='liblinear', random_state=1))
```

```
In [ ]: pipe.fit(X_train, y_train)
        preds = pipe.predict(X_test)
        pipe.score(X_test, y_test)
```

```
In [ ]:
```

step attribute

```
In [ ]: # cancer
        pipe.fit(cancer.data, cancer.target)
        # "pca"
        components = pipe.named_steps["pca"].components_
        print("components.shape: {}".format(components.shape))
```

```
In [ ]: from sklearn.linear_model import LogisticRegression

        pipe = make_pipeline(StandardScaler(), LogisticRegression())
```

```
In [ ]: param_grid = {'logisticregression__C': [0.01, 0.1, 1, 10, 100]}
```

```
In [ ]: from sklearn.model_selection import GridSearchCV
        X_train, X_test, y_train, y_test = train_test_split(
            cancer.data, cancer.target, random_state=4)
        grid = GridSearchCV(pipe, param_grid, cv=5)
        grid.fit(X_train, y_train)
```

```
In [ ]: print(" :\n{}".format(grid.best_estimator_))
```

```
In [ ]: print("  :\n{}".format(
              grid.best_estimator_.named_steps["logisticregression"]))
```

```
In [ ]: print("  :\n{}".format(
              grid.best_estimator_.named_steps["logisticregression"].coef_))
```

```
In [ ]: from sklearn.datasets import load_boston
        boston = load_boston()
        X_train, X_test, y_train, y_test = \
                      train_test_split(boston.data, boston.target,
                                                   random_state=0)

        from sklearn.preprocessing import PolynomialFeatures
        pipe = make_pipeline(
            StandardScaler(),
            PolynomialFeatures(),
            Ridge())
```

```
In [ ]: param_grid = {'polynomialfeatures__degree': [1, 2, 3],
                      'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
```

```
In [ ]: grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1)
        grid.fit(X_train, y_train)
```

```
In [ ]: import mglearn
        mglearn.tools.heatmap(grid.cv_results_['mean_test_score'].reshape(3, -1),
                              xlabel="ridge__alpha", ylabel="polynomialfeatures__degree",
                              xticklabels=param_grid['ridge__alpha'],
                              yticklabels=param_grid['polynomialfeatures__degree'], vmin=0)
```

```
In [ ]: print(" : {}".format(grid.best_params_))

In [ ]: print("  : {:.2f}".\
              format(grid.score(X_test, y_test)))

In [ ]: param_grid = {'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
        pipe = make_pipeline(StandardScaler(), Ridge())
        grid = GridSearchCV(pipe, param_grid, cv=5)
        grid.fit(X_train, y_train)
        print("    : {:.2f}".\
              format(grid.score(X_test, y_test)))
```

0.1.3 Model Selection

```
In [ ]: pipe = Pipeline([('preprocessing', StandardScaler()), \
                          ('classifier', SVC())])

In [ ]: from sklearn.ensemble import RandomForestClassifier

        param_grid = [
            {'classifier': [SVC()], 'preprocessing': [StandardScaler()],
             'classifier__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
             'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100]},
            {'classifier': [RandomForestClassifier(n_estimators=100)],
             'preprocessing': [None], 'classifier__max_features': [1, 2, 3]}]

In [ ]: X_train, X_test, y_train, y_test = train_test_split(
            cancer.data, cancer.target, random_state=0)

        grid = GridSearchCV(pipe, param_grid, cv=5)
        grid.fit(X_train, y_train)

        print(" :\n{}\n".format(grid.best_params_))
        print("   : {:.2f}".format(grid.best_score_))
        print("  : {:.2f}".format(grid.score(X_test, y_test)))
```