Week 4

Polynomial Regression

💡 Polynomial regression = Polynomial transformation + Linear Regression

Polynomial Features

- Polynomial features are those features created by raising existing features to an exponent.

- The “*degree*” of the polynomial is used to control the number of features added, e.g. a degree of 3 will add two new variables for each input variable.
- PolynomialFeatures transformer transforms an input data matrix into a new data matrix of a given degree.

Training

```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

# Two steps:
# 1. Polynomial features of the desired degree (here degree=2)
# 2. Linear regression

poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('linear_regression', LinearRegression())])

# Train with feature matrix X_train and label vector y_train
poly_model.fit(X_train, y_train)
```

Interaction Features

- It is also common to add new variables that represent the interaction between features, e.g a new column that represents one variable multiplied by another. This too can be repeated for each input variable creating a new “*interaction*” variable for each pair of input variables.
- The “*interaction_only*” argument means that only the raw values (degree 1) and the interaction (pairs of values multiplied with each other) are included.

```
from sklearn.preprocessing import PolynomialFeatures
poly_transform = PolynomialFeatures(degree=2, interaction_only=True)
```

Hyperparameter Tuning

- Hyper-parameters are parameters that are not directly learnt within estimators.
- In sklearn, they are passed as arguments to the constructor of the estimator classes.

Setting hyperparameters

Select hyperparameters that result in the best cross-validation scores.

Hyperparameter search consists of

- an estimator (regressor or classifier);
- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a score function.

Two approaches for Hyperparameter Tuning in SkLearn are:

- GridSearchCV
- RandomisedSearchCV

GridSearchCV

exhaustively considers all parameter combinations for specified values.

RandomisedSearchCV

samples a given number of candidate values from a parameter space with a specified distribution.

Steps in Hyperparameter Tuning

1. Divide data into training, validation and test sets.
2. For each combination of hyper-parameter values learn a model with a training set.
    a. This **step can be run in parallel by setting n_jobs = -1**.

b. Some parameter combinations may cause failure in fitting one or more folds of data. This may cause the search to fail. **Set error_score = 0 (or np.NaN) to set score for the problematic fold to 0 and complete the search**.

3. Evaluate the performance of each model with a validation set and select a model with the best evaluation score.
4. Retrain model with the best hyper-parameter settings on training and validation set combined.
5. Evaluate the model performance on the test set.

Regression Model-specific Hyperparameter Tuning

- Some models can fit data for a range of values of some parameter almost as efficiently as fitting the estimator for a single value of the parameter.
- This feature can be leveraged to perform more efficient cross-validation used for model selection of this parameter.

💡 linear_model.LassoCV

💡 linear_model.RidgeCV

💡 linear_model.ElasticNetCV

Regularization

Performing Ridge Regularisation

Option 1

1. Instantiate object of Ridge estimator
2. Set parameter alpha to the required regularization rate.

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1e-3)
```

Option 2

1. Instantiate object of SGDRegressor estimator

2. Set parameter alpha to the required regularization rate and penalty = l2.

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor(alpha=1e-3, penalty='l2')
```

Searching the best regularization parameter for Ridge

Option 1

Search for the best regularization rate with built-in cross validation in RidgeCV estimator.

Option 2

Use cross validation with Ridge or SVDRegressor to search for best regularization using

- Grid search

- Randomized search

Ridge regularization in polynomial regression

Set up a pipeline of polynomial transformation followed by the ridge regressor.

```
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
        ('polynomial_transform', PolynomialFeatures(degree=2)),
        ('ridge', Ridge(alpha=1e-3))])
poly_model.fit(X_train, y_train)
```

Instead of Ridge, we can use SGDRegressor to get equivalent formulation.

Performing Lasso Regularisation

Option 1

1. Instantiate object of Lasso estimator

2. Set parameter alpha to the required regularization rate.

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=1e-3)
```

Option 2

1. Instantiate object of SGDRegressor estimator
2. Set parameter alpha to the required regularization rate and penalty = l1.

```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor(alpha=1e-3, penalty='l1')
```

Searching the best regularization parameter for Lasso

Option 1

Search for the best regularization rate with built-in cross validation in LassoCV estimator.

Option 2

Use cross validation with Lasso or SVDRegressor to search for best regularization using

- Grid search
- Randomized search

Lasso regularization in polynomial regression

Set up a pipeline of polynomial transformation followed by the lasso regressor.

```
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
        ('polynomial_transform', PolynomialFeatures(degree=2))),
        ('lasso', Lasso(alpha=1e-3))])
poly_model.fit(X_train, y_train)
```

Instead of Lasso, we can use SGDRegressor to get equivalent formulation.

Perform both lasso and ridge regularization in polynomial regression

Set up a pipeline of polynomial transformation followed by the SGDRegressor with **penalty = ‘elasticnet’**

```
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([
                ('polynomial_transform', PolynomialFeatures(degree=2)),
                ('elasticnet', SGDRegressor(penalty='elasticnet',
                l1_ratio=0.3))])

poly_model.fit(X_train, y_train)
```

- Elasticnet is a convex combination of L1 (Lasso) and L2 (Ridge) regularization.
- In this example, we have set *l1_ratio* to *0.3*, which means *l2_ratio = 1- l1_ratio = 0.7* L2 takes higher weightage in this formulation.