Lernverfahren autonomer Roboter - Übung 7

Tobias Hahn
3073375

December 21, 2016

Übung 7

1 Normal Equation

$$
\begin{aligned}
X &= \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \\ 1 & 5 \\ 1 & 8 \end{bmatrix} \\
y &= \begin{bmatrix} 3.5 \\ 2.5 \\ 5.5 \\ 9.5 \\ 14.5 \end{bmatrix}
\end{aligned} \tag{1}
$$

$$
X^t X = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 5 & 8 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \\ 1 & 5 \\ 1 & 8 \end{bmatrix} = \begin{bmatrix} 5 & 20 \\ 20 & 110 \end{bmatrix} \tag{2}
$$

$$
(X^t X)^{-1} = \begin{bmatrix} 5 & 20 \\ 20 & 110 \end{bmatrix}^{-1} = \frac{1}{550 - 400} * \begin{bmatrix} 110 & -20 \\ -20 & 5 \end{bmatrix} = \begin{bmatrix} \frac{11}{15} & -\frac{2}{15} \\ -\frac{2}{15} & \frac{1}{30} \end{bmatrix} \tag{3}
$$

$$
(X^t X)^{-1} X^t = \begin{bmatrix} \frac{11}{15} & -\frac{2}{15} \\ -\frac{2}{15} & \frac{1}{30} \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 5 & 8 \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & \frac{7}{15} & \frac{1}{15} & \frac{1}{15} & -\frac{1}{3} \\ \frac{1}{10} & -\frac{1}{15} & 0 & \frac{1}{30} & \frac{2}{15} \end{bmatrix} \tag{4}
$$

$$
\hat{\omega} = (X^t X)^{-1} X^t y = \begin{bmatrix} \frac{3}{5} & \frac{7}{15} & \frac{1}{15} & \frac{1}{15} & -\frac{1}{3} \\ \frac{1}{10} & -\frac{1}{15} & 0 & \frac{1}{30} & \frac{2}{15} \end{bmatrix} * \begin{bmatrix} 3.5 \\ 2.5 \\ 5.5 \\ 9.5 \\ 14.5 \end{bmatrix} = \begin{bmatrix} -\frac{17}{30} \\ \frac{73}{30} \end{bmatrix} \tag{5}
$$

2 Normal Equation

Folgend der Code und die Ausgaben des Programms:

2.1 Code

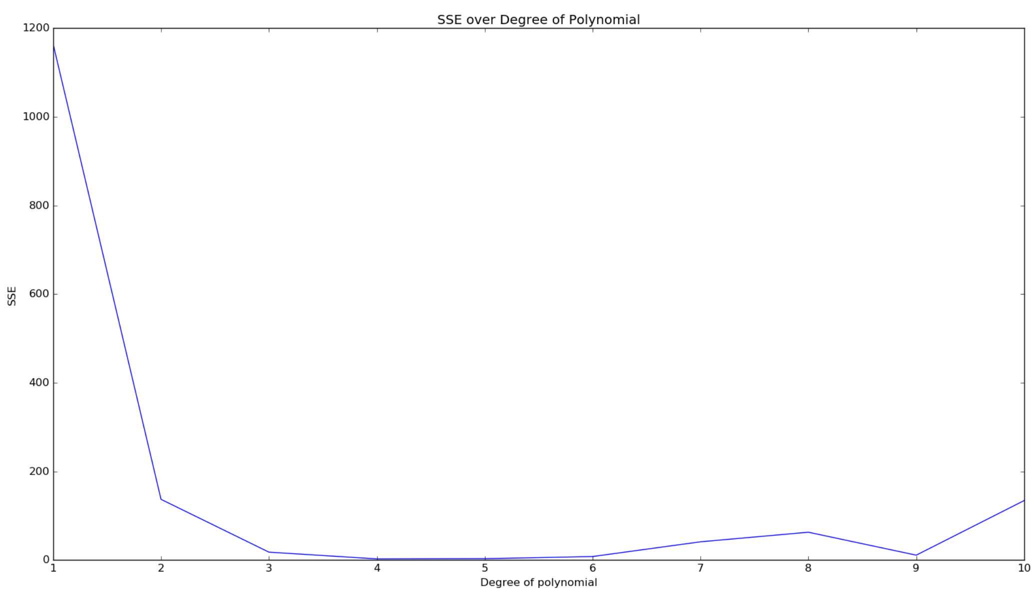
../code/regression.py

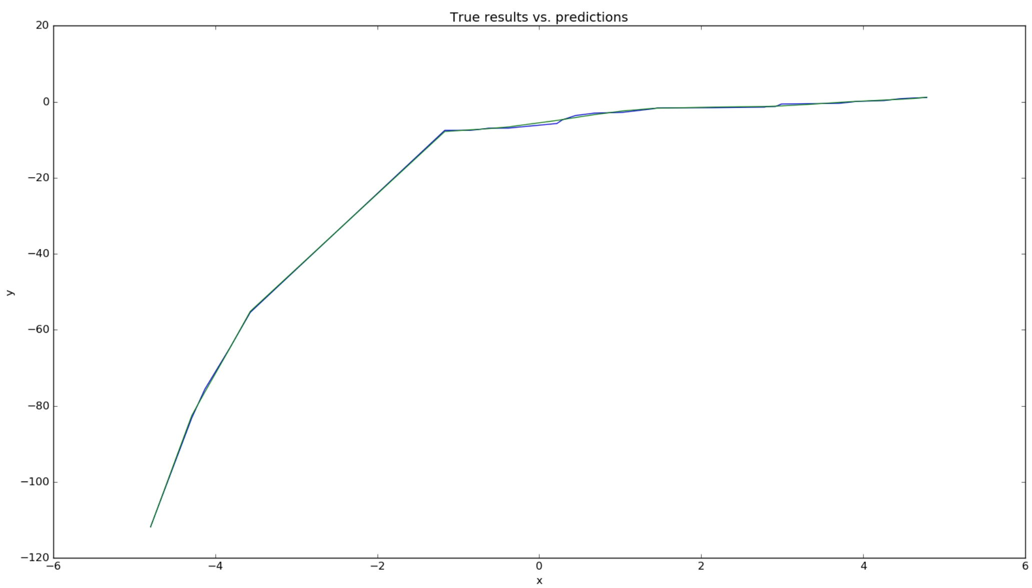
```python
import numpy as np
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt


class Regression:
    """This class holds different algorithms for regression
       and the cv function
    """
    def apply_k_fold_cv(self, X, y, regressor=None, n_folds=5, **kwargs):
        """K fold cross validation.

        Parameters
        ----------
        X : array-like, shape (n_samples, feature_dim)
            The data for the cross validation

        y : array-like, shape (n_samples, label_dim)
            The labels of the data used in the cross validation

        regressor : function
            The function that is used for regression of the training data

        n_splits : int, optional (default: 5)
            The number of folds for the cross validation

        kwargs :
            Further parameters that get used e.g. by the regressor

        Returns
        -------
        errors : array, shape (n_splits,)
            Vector of regression errors for the n_splits folds.
        """
        assert X.shape[0] == y.shape[0]

        if len(X.shape) < 2:
            X = np.atleast_2d(X).T
        if len(y.shape) < 2:
            y = np.atleast_2d(y).T

        cv = KFold(n_splits=n_folds, shuffle=True, random_state=42)
        errors = []

        for train_index, test_index in cv.split(X):
            train_data = X[train_index, :]
            train_label = y[train_index, :]
            test_data = X[test_index, :]
            test_label = y[test_index, :]
            error = regressor(train_data, test_data,
                              train_label, test_label, **kwargs)

            errors.append(error)

        return np.array(errors)

    def normal_equations(self, x, y, d=1, **kwargs):
        """Calculates a hypothesis using the training data X and y
           for a given polynomial degree d

        Parameters
        ----------
        x : array-like, shape (n_samples, 1)
            The data for the regression

        y : array-like, shape (n_samples, 1)
            The value of the function f(X) with possible noise
```

```

        d : int, optional (default: 1)
            The degree of the polynomial

        Returns
        -------
        w : array-like, shape (d+1, 1)
            Weights of the computated hypothesis model for the regression

        """
        X = self.phi(x, d)
        return np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(y)

    def ne_regressor(self, X_train, X_test, y_train, y_test, **kwargs):
        """Calculates the squared sum of errors for a hypothesis
           using an evaluation set

        Parameters
        ----------
        X_train : array-like, shape (n_samples, feature_dim)
            The data for the training of the regressor

        X_test : array-like, shape (n_samples, feature_dim)
            The data for the test of the regressor

        y_train : array-like, shape (n-samples, label_dim)
            The labels for the training of the regressor

        y_test : array-like, shape (n-samples, label_dim)
            The labels for the test of the regressor

        Returns
        -------
        sse : double
            Sum of squared errors for the hypothesis using the evaluation set
        """

        X = self.phi(X_test, **kwargs)
        res = X.dot(self.normal_equations(X_train, y_train, **kwargs))
        return np.sum((y_test - res)**2)

    def phi(self, x, d=1):
        """Transforms a value x into a vector with components up to
           polynomial of degree d

        Parameters
        ----------
        X : array-like, shape (n_samples, 1)
            The data to be transformed

        d : int, optional (default: 1)
            The degree of the polynomial

        Returns
        -------
        transformed_values : array-like, shape (n_samples, d+1)
            The polynomial values of the input value(s) x
        """

        X = np.ones((x.shape[0], d+1))
        for i in range(1, d+1):
            X[:, i] = (x**i).flatten()
        return X

if __name__ == '__main__':
    # Instance of the Regression class holding regression algorithms
    r = Regression()

    ### YOUR IMPLEMENTATION FOR EXERCISE 2 (b) GOES HERE ###
    res = np.sort(np.loadtxt('../data/data.txt'))
    x = np.atleast_2d(res[0,:]).T
```

```
    y = np.atleast_2d(res[1,:]).T


    dimensions = range(1,11)
    errors = [np.mean(r.apply_k_fold_cv(x, y, r.ne_regressor, 10, d=dim)) for dim in
        dimensions]

    best_dim = np.argmax(dimensions)+1
    X = r.phi(x, d=best_dim)
    res = X.dot(r.normal_equations(x, y, d=best_dim))

    plt.xlabel('Degree of polynomial')
    plt.ylabel('SSE')
    plt.title('SSE over Degree of Polynomial')
    plt.plot(dimensions, errors)
    plt.show()

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('True results vs. predictions')
    plt.plot(x, y, x, res)
    plt.plot()
    plt.show()
```

2.2 Results




True results vs. predictions
20
0
−20
−40
−60
−80
−100
−120
y
−6
−4
−2
0
2
4
6
x