Model Assessment and Selection II

Shizhi Chen-10307389

Assignment 1:

Complete the functions aic_loss and bic_loss for calculating the AIC and the BIC scores for a learned or trained polynomial model. In your implementation, you can use the provided function to calculate the degree of a given polynomial model. Run the code and record the RSS, AIC, and BIC scores on the given data set. Repeat this for polynomial models of degrees 1, 3, and 5. Report all the scores for polynomial models of different degrees. With the results achieved, comment on which polynomial degree has been used to generate the data set.

Python Code:

```
# Load the libraries needed
%matplotlib inline
import csv
import numpy as np
import matplotlib.pyplot as plt
import time

# Seed random number generator
np.random.seed(123)

# Helper functions for polynomials
def powers( X, n ):
    """ Returns an array of the powers of the elements of X up to the nth power """
    return np.power(np.expand_dims(X, axis=-1), [np.arange(n)])

def polynomial( X, poly_coeff ):
    """ Returns the value of a polynomial at x with given coefficients """
    deg = np.shape(poly_coeff)[-1]
    return np.dot(powers(X, deg), np.transpose(poly_coeff), )

def fit_polynomial( X, y, n ):
    """ Returns the coefficients of the n-degree polynomial fit to (X, y) """
    X_pwrs = powers(X, n+1)
    # Do linear least squares fit
    coeff, _, _, _ = np.linalg.lstsq(X_pwrs, y, rcond=None)
    return coeff

def degree( poly_coeff ):
    """ Returns the degree of a polynomial from its coefficients """
    return len(poly_coeff)-1
```

First, modify the functions aic_loss and bic_loss.

$$\text{AIC} = \frac{1}{n}\left[\text{RSS} + 2d(\Theta)\right], \quad \text{BIC} = \frac{1}{n}\left[\text{RSS} + d(\Theta)log(n)\right] \text{ where RSS} = \sum_{i=1}^{n}\left[y_i - \hat{f}(\boldsymbol{x}_i, \Theta)\right]^2$$

Thus, we can modify the function as following:

```
# RSS, AIC, and BIC scores
def rss_loss(X, y, learned_coeff):
    """ Computes the residual sum of squares loss for a given polynomial on the given data """
    pred_y = polynomial(X, learned_coeff)
    return np.sum(np.square(pred_y - y))

def aic_loss(X, y, learned_coeff):
    """ Computes the Akaike's Information Criterion for a given polynomial on the given data """
    # You will need to modify this function to return the correct result
    aic=(rss_loss(X, y, learned_coeff)+2*(len(learned_coeff)))/len(X)
    return aic

def bic_loss(X, y, learned_coeff):
    """ Computes the Bayesian Information Criterion for a given polynomial on the given data """
    # You will need to modify this function to return the correct result
    bic=(rss_loss(X, y, learned_coeff)+np.log(len(X))*(len(learned_coeff)))/len(X)
    return bic
```

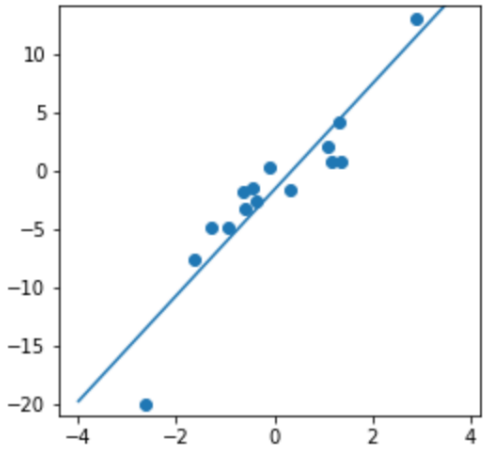
Then, compare the polynomial models of degrees 1, 3, and 5.

```
# Load data
X, y = np.load("./class9_generated_data.npy")

#Plot the data
plt.figure(figsize=(4,4))
plt.scatter(X,y)
plt.ylim(y.min()-1., y.max()+1.)

# Make sequence of x-values for plotting
axis_X = np.arange(-4.0, 4.0, 0.2)

# Fit a polynomial
n = 1
learned_coefficients = fit_polynomial( X, y, n )

#Plot the learned polynomial
poly_y = polynomial(axis_X, learned_coefficients)
plt.plot(axis_X, poly_y )

plt.show()

num_points = np.shape(X)[0]
rss = rss_loss(X, y, learned_coefficients)
aic = aic_loss(X, y, learned_coefficients)
bic = bic_loss(X, y, learned_coefficients)
str_ = "Fitted a degree {:d} polynomial with ".format(degree(learned_coefficients))
str_ += "RSS loss: {:.2f}, AIC loss: {:.2f}, BIC loss: {:.2f} ".format(rss, aic, bic)
print(str_ + "to {:d} datapoints.".format(num_points))
```

change the degree here

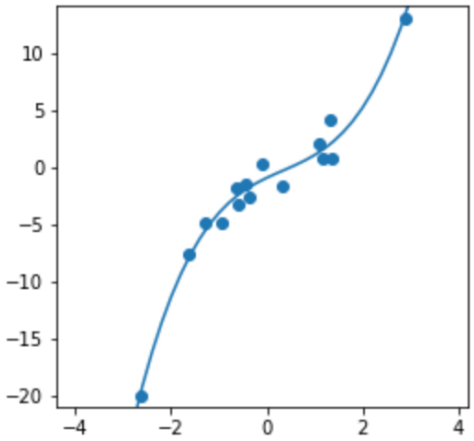
Results:

Degree=1



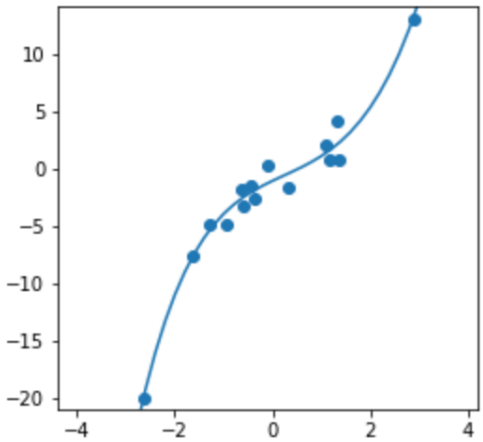
Fitted a degree 1 polynomial with RSS loss: 100.73, AIC loss: 6.98, BIC loss: 7.08 to 15 datapoints

Degree=3



Fitted a degree 3 polynomial with RSS loss: 15.57, AIC loss: 1.57, BIC loss: 1.76 to 15 datapoints

Degree=5



Fitted a degree 5 polynomial with RSS loss: 15.34, AIC loss: 1.82, BIC loss: 2.11 to 15 datapoints

Comment:

Compared with the models with different degrees, we find that the degree 1 model has the highest RSS, AIC and Bic loss so the degree 1 model is considered as underfitted. As for degree 5 model, which has the lowest RSS loss but higher AIC and BIC loss than degree 3 model. The model with degree 3 has the lowest AIC and BIC loss, and its RSS loss is just slightly higher than degree 5 but also relatively low. Therefore, the degree 3 can be a good polynomial degree used to generate data set.

Assignment 2: Modify the provided forward feature selection routine to use the AIC score instead of the RSS as its evaluation criterion. Run your modified code on the Automobile data set to select a best feature subset for the insurance category prediction of a vehicle.

Python Code:

```
# Helper functions for linear models
def linear ( X, coefficients ):
    return np.dot(X, np.transpose(coefficients), )

def fit_linear( X, y, features=None ):
    """
        Returns the coefficients of a linear model fit to X,y.
        If features is a list of integers, then fit will ignore
        any features whose index is not in the list.
        ( Returned coefficients for these features will be set
        to 0. )
    """
    if features is not None:
        # Make a mask
        tot_num_features = np.shape(X)[-1]
        mask = np.zeros((1,tot_num_features))
        mask[0,features] = 1.
        # Zero out all irrellevant features
        X = X * mask

    # Do linear least squares fit
    coeff, _, _, _ = np.linalg.lstsq(X, y, rcond=None)
    return coeff
```

```
# Load data
data = []
continuous_features = [ 0, 1, 9, 10, 11, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 25 ]
# Original data is from https://archive.ics.uci.edu/ml/datasets/automobile
```

```
data = np.array(data)
y = data[:, 0] # target is first value
X = data[:, 1:] # training data is the rest

# Normalize the data to zero mean and unit std
X = (X - np.mean(X, axis=0, keepdims=True)) / np.std(X, axis=0, keepdims=True)
y = (y - np.mean(y)) / np.std(y)

print("Sucessfully loaded {:d} entries.\n".format(len(X)))


# Forwards stepwise feature selection
tot_num_features = np.shape(X)[-1]
all_features = range(tot_num_features)
curr_features = []

best_score = 1e9
best_features = None

print("Performing forwards stepwise feature selection using AIC as criteria...")
```

```
while len(curr_features) != tot_num_features:
    # Get remaining features
    candidate_features = [ f for f in all_features if f not in curr_features ]
    best_score_this_round = 1e9
    best_feature_this_round = None
    for f in candidate_features:
        test_features = curr_features + [f]
        learned_coefficients = fit_linear( X, y, test_features )

        y_pred = linear(X, learned_coefficients)
        rss=np.sum(np.square(y_pred - y))
        aic = (rss+ 2*(len(test_features)+1))/len(X)
        score = aic

        # Remember, lower score is better
        if score < best_score_this_round:
            best_score_this_round = score
            best_feature_this_round = f
        if score < best_score:
            best_score = score
            best_features = test_features
    # Set current features to best features from round
    curr_features = curr_features + [best_feature_this_round]
    print("Round {}, selected feature {:d} with score {:.3f}"\
          .format(len(curr_features), best_feature_this_round, best_score_this_round ))
print("Best features were {} ({:d} total) with score {:.3f}".format(best_features, len(best_features), best_score))
```

Use AIC score

```
Sucessfully loaded 160 entries.

Performing forwards stepwise feature selection using AIC as criteria...
Round 1, selected feature 1 with score 0.754
Round 2, selected feature 0 with score 0.530
Round 3, selected feature 3 with score 0.490
Round 4, selected feature 2 with score 0.498
Round 5, selected feature 9 with score 0.506
Round 6, selected feature 12 with score 0.516
Round 7, selected feature 13 with score 0.517
Round 8, selected feature 4 with score 0.528
Round 9, selected feature 10 with score 0.539
Round 10, selected feature 5 with score 0.550
Round 11, selected feature 11 with score 0.560
Round 12, selected feature 6 with score 0.572
Round 13, selected feature 7 with score 0.584
Round 14, selected feature 14 with score 0.596
Round 15, selected feature 8 with score 0.608
Best features were [1, 0, 3] (3 total) with score 0.490
```

From the forward stepwise feature selection results, we can see that the features 1, 0 and 3 make the model obtains the best AIC score 0.490.

Assignment 3: Implement the backward stepwise feature selection routine. Run your backward selection code on the Automobile data set to select a best feature subset for the insurance category prediction of a vehicle. Compare the results of running backwards feature selection to running forwards feature selection. With the results and your justification, comment on whether there is a difference between two feature selection methods in terms of this data set.

Algorithm about backward stepwise selection:

- Backward stepwise selection

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   - 2.1 Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   - 2.2 Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Python Code:

```
# Backwards feature selection
curr_features = list(range(X.shape[-1]))

learned_coefficients=fit_linear(X,y,curr_features)
y_pred = linear(X, learned_coefficients)
rss=np.sum(np.square(y_pred - y))
aic_score = (rss + 2*(len(test_features)+1))/len(X)

best_score = 1e9
best_features=None

print("Performing backwards stepwise feature selection using AIC as criteria...")

for no_round in range(1,X.shape[-1]):

    best_score_this_round=1e9
    worst_feature_this_round=None
```

```
    for i in range(len(curr_features)):
        f=curr_features[i]
        test_features=curr_features
        test_features=list(filter(lambda x:x!=f,test_features))

        learned_coefficients=fit_linear(X,y,test_features)
        y_pred = linear(X, learned_coefficients)
        rss=np.sum(np.square(y_pred - y))
        aic = (rss + 2*(len(test_features)+1))/len(X)

        if aic < best_score_this_round:
            worst_feature_this_round = f
            best_score_this_round = aic
        if aic < aic_score:
            best_features = test_features
            aic_score = aic
    # Set current features to best features from round
    curr_features = list(filter(lambda x:x!=worst_feature_this_round,curr_features))
    print("Round {}, deleted feature {:d} and get the score {:.3f}"\
          .format(X.shape[-1]-len(curr_features)+1, worst_feature_this_round, best_score_this_round ))
print("Best features were {} ({:d} total) with score {:.3f}".format(best_features, len(best_features), aic_score))
```

```
Sucessfully loaded 160 entries.

Performing backwards stepwise feature selection using AIC as criteria...
Round 2, deleted feature 8 and get the score 0.596
Round 3, deleted feature 14 and get the score 0.584
Round 4, deleted feature 2 and get the score 0.571
Round 5, deleted feature 7 and get the score 0.559
Round 6, deleted feature 6 and get the score 0.548
Round 7, deleted feature 11 and get the score 0.538
Round 8, deleted feature 5 and get the score 0.527
Round 9, deleted feature 10 and get the score 0.516
Round 10, deleted feature 4 and get the score 0.505
Round 11, deleted feature 9 and get the score 0.502
Round 12, deleted feature 13 and get the score 0.502
Round 13, deleted feature 12 and get the score 0.490
Round 14, deleted feature 3 and get the score 0.530
Round 15, deleted feature 0 and get the score 0.754
Best features were [0, 1, 3] (3 total) with score 0.490
```

Comment:

In this data set, backward stepwise feature selection method shows the same result as forwards stepwise feature selection. The best features are 0,1,3 with a score of 0.490.

Like forward stepwise selection, the backward selection approach searches through only 1+p (p+ 1)/2 models. However, backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit). In contrast, forward stepwise can be used even when n<p, and so is the only viable subset method when p is large.