Machine Learning: Basic Principles

Model Validation and Selection

Learning goals

Remember the main componets of a machine learning problem:

1. **data points** which are characterized by features $\mathbf{x}$ and labels y,
2. a **hyopothesis space** $\mathcal{H}$ collecting feasible maps (predictors) $h(\mathbf{x})$ from features $\mathbf{x}$ to a predicted label $\hat{y} = h(\mathbf{x})$ and
3. a **loss function** (such as squared error loss or the logistic loss).

In this exercise you will learn a principled approach for **model selection**, i.e., how to optimally choose the best (in a certain sense) hypothesis space $\mathcal{H}$ from a whole ensemble of different hpyothesis spaces $\mathcal{H}^{(1)}, \mathcal{H}^{(2)}, \ldots$. This approach is based on computing, for each hypothesis space $\mathcal{H}^{(d)}$, the empirical risk (the validation error) of the predictor $\hat{h}$ obtained by **empirical risk minization** over $\mathcal{H}^{(d)}$. The data points used for computing the validation error is known as **validation set** or **cross-validation set** $\mathcal{X}^{(\text{val})}$.

The validation set should contain data points which have not already been used for choosing the predictor out of $\mathcal{H}^{(d)}$ via empirical risk minimization. We can then compare the quality of different models by comparing the validation error incurred by the predictors that have been obtained using the different models.

We will also learn about **regularization** as a "soft" variant of model selection. Here, we use a fixed but large hypothesis space (e.g., space of polynomials with a large degree) and regularize the empirical risk minimization problem by adding a term that quanties the "complexity" of a particular predictor (e.g., a polynomial of high degree).

The concept of model selection and regularization is best understood by working through a particular example. We will use as this example the problem of predicing cryptocurrencies which you already faced in Round 2 - "Regression".

Exercise Content

1. Predicting Cryptocurrencies
2. Model Selection
3. Regularization

Keywords

`Regularization`, `Validation`, `Model Selection`, `Polynomial Regression', 'Regularized Empirical Risk Minimization'

1. Predicting A Cryptocurrency

We consider again (see Round 2 "Regression") the problem of predicting the closing price of the cryptocurrency Ethereum based on the closing price of Bitcoin on the same day. However, we will only consider a **subset** of the historic data about Ethereum and Bitcoin used in round `2.Regression`. This subset of $N = 20$ data points $\mathcal{X} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ is stored in the file *BTC_ETH_round4.csv*.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display

#Read the data
df=pd.read_csv('BTC_ETH_round4.csv')
x=df.Bitcoin.values #Bitcoin values
y=df.Ethereum.values #Ethereum values

#Reshape the data.
#This is really important for the matrix multiplication later on!
x=np.reshape(x,(len(x),1))
y=np.reshape(y,(len(y),1))

#Plot the data
plt.figure(figsize=(6, 6))
plt.scatter(x, y)
plt.title(r'$\bf{Figure\ 1.}$ Bitcoin vs Ethereum')
plt.xlabel('Bitcoin')
plt.ylabel('Ethereum')
plt.show()
print(len(x))
```

```
<Figure size 600x600 with 1 Axes>
```

```
20
```

2 Model Selection

As observed in Round 2 ("Regression"), the relation between the Ethereum price y and the Bitcoin price x is highly non-linear. Therefore it is useful to consider a hypothesis space which is constituted by polynomial functions of degree d:

$$\mathcal{H}^{(d)}_{\text{poly}} = \{h^{(\mathbf{w})}(\cdot) : \mathbb{R} \to \mathbb{R} : h^{(\mathbf{w})}(x) = \sum_{r=1}^{d+1} w_r x^{r-1}, \text{ with some } \mathbf{w} = (w_1, \ldots, w_{d+1})^T \in \mathbb{R}^{d+1}\}.$$

As discussed in the course book (Section 3.2 "Polynomial Regression"), polynomial regression is equivalent to combining linear regression with a feature map. In particular, we transform the features x (Bitcoin closing price) of the data points to a higher dimensional feature space using the feature map

$$\phi(x) = (x^0, \ldots, x^{d-1})^T \in \mathbb{R}^d. \tag{7}$$

Using this new feature vector $\mathbf{x} = \phi(x)$, we can represent any polynomial map $h \in \mathcal{H}^{(d)}_{\text{poly}}$ as

$$h^{(\mathbf{w})}(x) = \mathbf{w}^T \phi(x) \text{ with some weight vector } \mathbf{w} \in \mathbb{R}^{d+1}. \tag{8}$$

Consider a particular choice for the maximum degree (e.g., $d = 2$). We can find a good polynomial predictor $\hat{h}^{(d)}$ by minimizing the empirical risk over some labeled datapoints (the training set). However, we will not use the entire dataset $\mathcal{X}$ for this empirical risk minimization but only a subset, the training set

$$\mathcal{X}^{(\text{train})} = \{(x^{(2)}, y^{(2)}), (x^{(4)}, y^{(4)}), (x^{(6)}, y^{(6)}), \ldots, (x^{(N)}, y^{(N)})\}.$$

This training set $\mathcal{X}^{(\text{train})}$, of size $N_{\text{train}} = N/2$, is obtained by selecting every other data point of the original data set $\mathcal{X}$. The remaining data points in $\mathcal{X}$ will be used as the validation set:

$$\mathcal{X}^{(\text{val})} = \{(x^{(1)}, y^{(1)}), (x^{(3)}, y^{(3)}), (x^{(19)}, y^{(19)}).$$

For each choide of d, which corresponds to a particular hypothesis space $\mathcal{H}^{(d)}$, we learn the optimal predcitor $\hat{h}^{(d)}$ by solving

$$\hat{h}^{(d)} = \arg \min_{h \in \mathcal{H}^{(d)}_{\text{poly}}} (1/N_{\text{train}}) \sum_{i=2,4,\ldots,20} (y^{(i)} - h(x^{(i)})^2. \tag{9}$$

This optimization problem can be rewritten using the paramtrization (7) as

$$\hat{\mathbf{w}}^{(d)} = \arg \min_{\mathbf{w} \in \mathbb{R}^{d+1}} (1/N_{\text{train}}) \sum_{i=2,4,\ldots,20} \left(y^{(i)} - \mathbf{w}^T \phi\left(x^{(i)}\right)\right)^2. \tag{10}$$

Given the optimal weight vector $\hat{\mathbf{w}}^{(d)}$ which solves (10), we obtain the optimal predictor $\hat{h}^{(d)}$ (which solves (9)) by $\hat{h}^{(d)}(x) = \left(\hat{\mathbf{w}}^{(d)}\right)^T \phi(x)$.

This optimization problem can solved either using a closed-form expression (under certain conditions) involving the feature matrix $\mathbf{X} = \left(\phi\left(x^{(2)}\right), \phi\left(x^{(4)}\right), \ldots, \phi\left(x^{(20)}\right)\right)^T$ or using gradient descent (with a sufficiently small step size). Let us denote the training error, which is the minimum objective value in (9) and (10), obtained for degree d as $E^{(d)}_{\text{train}}$.

After finding the predictor $\hat{h}^{(d)}$ for different choices of the maximum degree d, we evaluate the empirical risk incurred by $\hat{h}^{(d)}$ over the validation set:

$$E_{\rm val}^{(d)} = (1/N_{\rm val}) \sum_{i=1,3,\ldots,19} \left(y^{(i)} - \mathbf{w}^T \boldsymbol{\phi}\left(x^{(i)}\right)\right)^2.$$

We then choose the degree d for which the validation error $E_{\rm val}^{(d)}$ is smallest. Note that we do not use the training error $E_{\rm train}^{(d)}$ for choosing the degree d. In general it is not a good idea to choose the hypothesis space based on how small the training error is. This is because for very large hypothesis spaces, like polynomials with a large degree, we can always find a predictor from it which **by accident** is able to fit the training data well. However, such a predictor will perform poorly when applied to other data points, such as the data points in the validation set.

- The Python function `trainValErrors()` below takes as input the training data, validation data and a list of degrees. And gives as output 3 vectors which contain for every degree the optimal weight, *w_opts*, the training error, *training_errors* and the validation error, *validation_errors* :
  - Use the given functions, these are the same as the ones used for Polynomial Regression in round 2.
  - First get the optimal weights by only using the training data
  - Then compute the empirical risk separately for the training data and validation data

In [5]:

```
def fit(x, y):
    #Calculate the optimal weight vector
    w_opt = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(x), x)),np.transpose
(x)), y)
    return w_opt

def feature_mapping(x, degree):
    #compute the feature map for the given degree
    polynomial_features = np.column_stack([x**d for d in range(0,degree)])
    return polynomial_features

def polynomialRegression(x, y, degree):
    X = feature_mapping(x,degree)
    w_opt=fit(X,y)
    return w_opt

def predict(X, w_opt):
    #predict the labels
    y_pred = np.dot(X,w_opt)
    return y_pred

def empirical_risk(X, y, w_opt):
    empirical_error = np.mean(np.power(np.subtract(np.dot(X,w_opt), y),2))
    return empirical_error


#Split the data into a training and validation set.
#Don't change this.
x_train=x[1::2]
x_val=x[0::2]
y_train=y[1::2]
y_val=y[0::2]

#the degrees we want to loop over
degrees=[1,2,3,4,5,6]

def trainValErrors(x_train,y_train,x_val,y_val,degrees):
    ### STUDENT TASK ###
    #compute the optimal weight, training and validation error for each degree i
n degrees
    w_opts = []
    training_errors=[]
    validation_errors=[]
    for d in degrees:
        w_opt=polynomialRegression(x_train, y_train, d)
        w_opts.append(w_opt)
        training_errors.append(empirical_risk(feature_mapping(x_train,d),y_train
,w_opt))
        validation_errors.append(empirical_risk(feature_mapping(x_val,d),y_val,w
_opt))
    return w_opts, training_errors,validation_errors

#compute the training and validation errors and display them
w_opts, training_errors, validation_errors = trainValErrors(x_train,y_train,x_va
l,y_val,degrees)
df_degrees=pd.DataFrame(data={'d':degrees,'E_train':training_errors,'E_val':vali
dation_errors})
display(df_degrees)
```

```
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(121)

#plot the predictions for the different degrees
x_plot = np.linspace(0, 1, 50)

plt.scatter(x_train, y_train, color='black', label='Training data')
plt.scatter(x_val,y_val,color='red',label='Validation data')
for i in range(len(degrees)):
    plt.plot(x_plot, predict(feature_mapping(x_plot,degrees[i]),w_opts[i]), labe
l='degree = %d' %degrees[i])

plt.title(r'$\bf{Figure\ 2.}$ Polynomial Regression for different degrees')
plt.xlabel('Bitcoin')
plt.ylabel('Ethereum')
plt.ylim(0,1)
plt.xlim(0,1)
plt.legend()

#plot the training and validation errors for the different degrees
ax = fig.add_subplot(122)
plt.plot(degrees,training_errors,color='black',label='Training error')
plt.plot(degrees,validation_errors,color='red',label='Validation error')

plt.title(r'$\bf{Figure\ 3.}$ Training and validation error for different degree
s')
plt.ylabel('Empirical error')
plt.xlabel('Degree')
plt.xticks(degrees)
plt.legend()

plt.show()
```

	d	E_train	E_val
0	1	0.047725	0.085636
1	2	0.005044	0.018037
2	3	0.003882	0.013717
3	4	0.003613	0.021369
4	5	0.003581	0.013433
5	6	0.002970	1.346183

Figure 2. Polynomial Regression for different degrees

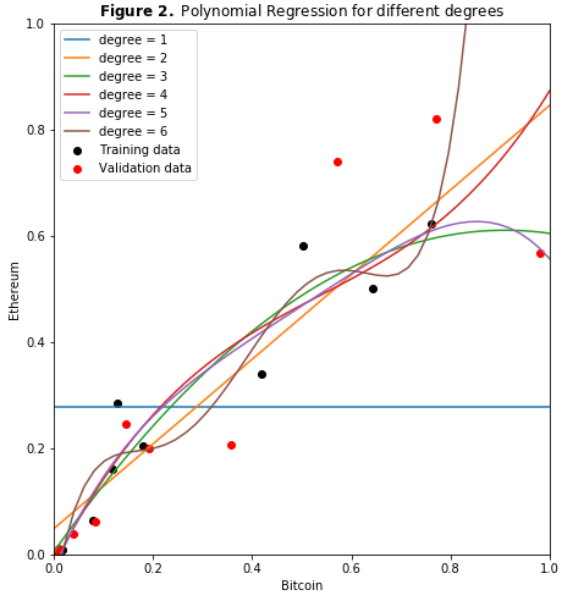
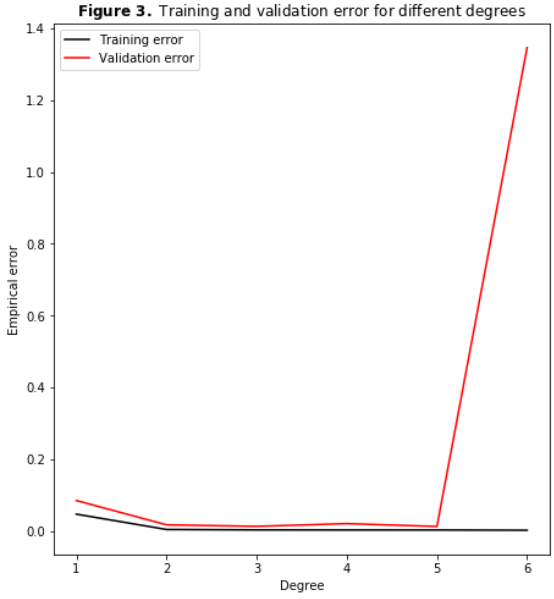


Figure 3. Training and validation error for different degrees



3 Regularization

In Section 2 we have tried to find the best hypothesis space within a sequence of increasing spaces $\mathcal{H}^{(1)} \subseteq \mathcal{H}^{(2)} \subseteq \ldots \subseteq \mathcal{H}^{(d_{\max})}$. Indeed, the space of polynomials with maximum degree $d = 1$ is a subset of the space of all polynomials with maximum degree $d = 2$. The simple but powerful idea behind model selection is to use the validation errors obtained for the different hypothesis spaces and picking the one resulting in the smallest validation error. An alternative approach to model selection is to use the largest hypothesis space $\mathcal{H}^{(d_{\max})}$ and to **regularize** the corresponding empirical risk minimization problem:

$$\hat{\mathbf{w}}^{(\lambda)} = \arg \min_{h \in \mathcal{H}^{(d_{\max})}} (1/N) \sum_{i=1}^{N} (y^{(i)} - \mathbf{w}^T \phi(x^{(i)})^2 + \lambda \|\mathbf{w}\|_2^2. \tag{11}$$

The function $\mathcal{R}(h)$ measures the **complexity** of a particular predictor map h. There are many different choices for this complexity function $\mathcal{R}(h)$ which mainly depend on the particular application at hand. Here, we will consider a particular choice which has proven useful for polynomial regression. To this end, note that in polynomial regression we can parametrize a predictor as $h^{(\mathbf{w})}(x) = \mathbf{w}^T \phi(x) = \sum_{l=0}^{d} w_{l+1} x^l$ with some weight vector $\mathbf{w} \in \mathbb{R}^{d+1}$. The entries w_l of the weight vector $\mathbf{w} = (w_1, \ldots, w_{d+1})^T$ are the cefficients of the powers $x^0, x^1, \ldots, x^d$ of the feature x (e.g., the Bitcoin closing price).

It is reasonable to measure the complexity of $h^{(\mathbf{w})}$ by the average size of the coeffcients w_l. Indeed, for many large coefficients w_l, the polynomial map $h^{(\mathbf{w})}(x)$ tends to be more ''wobbly''. In particular, we will use the regularizer $\mathcal{R}(h^{(\mathbf{w})}) = \|\mathbf{w}\|_2^2 = w_1^2 + \ldots + w_{d+1}^2$, which results in the following **regularized empirical risk minimization problem**

$$\hat{\mathbf{w}}^{(\lambda)} = \arg \min_{\mathbf{w} \in \mathbb{R}^{d+1}} (1/N_{\text{train}}) \sum_{i=2,4,\ldots,20} (y^{(i)} - \mathbf{w}^T \phi(x^{(i)}))^2 + \lambda \|\mathbf{w}\|_2^2. \tag{12}$$

The resulting weight vector $\hat{\mathbf{w}}^{(\lambda)}$ is then used to construct a predictor $\hat{h}^{(\lambda)}$ as

$$\hat{h}^{(\lambda)}(x) = \left(\hat{\mathbf{w}}^{(\lambda)}\right)^T \phi(x). \tag{13}$$

The constant $\lambda \geq 0$ is a tuning parameter and controls the effective degree of the rsulting predictor $\hat{h}^{(\lambda)}(x)$. If we choose a large value of λ, we will obtain a predictor $\hat{h}^{(\lambda)}(x)$ which resembles a polynomial of small degree (say, $d = 2$). In contrast, if we choose a very small value of λ, we will typically obtain a predictor $\hat{h}^{(\lambda)}$ which resembles a high-degree polynomial.

In a certain sense, we replace (or approximate) the computation of the validation error incurred by $h^{(\mathbf{w})}$ by adding the regularization term $\lambda \|\mathbf{w}\|_2^2$ to the training error. The term $\lambda \|\mathbf{w}\|_2^2$ in (12) accounts for the expected increase in validation error due to the complexity of the predictor $h^{(\mathbf{w})}(x) = \mathbf{w}^T \phi(x)$.

Similar to linear regression, we can derive a closed-form solution for the optimal weights in (12) as

$$\begin{aligned} \mathbf{w}_{\text{opt}} &= \text{argmin}_{\mathbf{w} \in \mathbb{R}^{d_{\max}+1}} \left[(1/N_{\text{train}}) \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \right] \\ &\Rightarrow \mathbf{w}_{\text{opt}} = (1/N_{\text{train}})((1/N_{\text{train}})\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}. \end{aligned} \tag{14}$$

Here, we used the feature matrix $\mathbf{X} = \left(\phi(x^{(2)}), \phi(x^{(4)}), \ldots, \phi(x^{(20)})\right)^T \in \mathbb{R}^{10 \times (d_{\max}+1)}$.

It remains to specify the choice of the regularization parameter λ in (12). One option is to try out different values for λ, compute the weight vector $\hat{\mathbf{w}}^{(\lambda)}$ by solving the problem (12) and evaluate the validation error of the corresponding predictor $h^{(\lambda)}$ (see (13)):

$$E_{\rm val}^{(\lambda)} = (1/N_{\rm val}) \sum_{i=1,3,\ldots,19} (y^{(i)} - \mathbf{w}^T \phi(x^{(i)}))^2.$$

We then pick the value for λ which results in the smallest validation error $E_{\rm val}^{(\lambda)}$.

Tasks

1. Implement a Python function `regularizedFit()` which takes as input $\mathbf{X}$, $\mathbf{y}$ and λ and returns the optimal weight vector according to (Eq. 14).
    - You can look at the numpy functions being used in `fit()` implemented in Section 2.
    - You can use `np.eye()` to generate an identity matrix
2. Implement a Python function `regularizedPolynomialRegression()` which takes as input $\mathbf{x}$, $\mathbf{y}$,λ and the degree. It should return the optimal weight vector.
    - Use `feature_mapping()` from Section 2 to get the feature matrix.
    - Use `regularizedFit()` to compute the optimal weight vector.
3. Implement a Python function `trainValErrorsRegularization()` which takes as inputs the training data, validation data, the list of lambdas and the degree. And gives as output 3 vectors which contain for every lambda the optimal weight, *w_opts*, the training error, *training_errors* and the validation error, *validation_errors*.
    - You can largely reuse `trainValErrors()` from Section 2. The difference is that you should now loop over the different lambda values, instead of the different degrees.

In [6]:

```
def regularizedFit(X, y,l=0):
    ### STUDENT TASK ###
    N=X.shape[0]
    w_opt = 1/N*np.dot(np.dot(np.linalg.inv(1/N*np.dot(np.transpose(X), X)+l*np.
eye(X.shape[1])),np.transpose(X)), y)
    return w_opt

def regularizedPolynomialRegression(x, y, l=0,degree=2):
    ### STUDENT TASK ###
    X = feature_mapping(x,degree)
    w_opt=regularizedFit(X,y,l)
    return w_opt

#specify the degree
degree=6
#specify list of values for lambda to be considered
lambdas = [0,0.01,0.5,1, 2, 5]

def trainValErrorsRegularization(x_train,y_train,x_val,y_val,lambdas=[0],degree=
2):
    ### STUDENT TASK ###
    #compute the optimal weight, training and validation error for each lambda
    w_opts = []
    training_errors=[]
    validation_errors=[]
    for l in lambdas:
        w_opt=regularizedPolynomialRegression(x_train, y_train, l,degree)
        w_opts.append(w_opt)
        training_errors.append(empirical_risk(feature_mapping(x_train,degree),y_
train,w_opt))
        validation_errors.append(empirical_risk(feature_mapping(x_val,degree),y_
val,w_opt))
    return w_opts, training_errors,validation_errors

#compute the training and validation errors and display them
w_opts_reg, training_errors_reg, validation_errors_reg = trainValErrorsRegulariz
ation(x_train,y_train,x_val,y_val,lambdas,degree=degree)
df_lambdas=pd.DataFrame(data={'Lambdas':lambdas,'Training errors':training_error
s_reg,'Validation errors':validation_errors_reg})
display(df_lambdas)

fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(121)
plt.scatter(x_train, y_train, color='black', label='Training data')
plt.scatter(x_val,y_val,color='red',label='Validation data')
x_plot = np.linspace(0, 1, 50)
for i in range(len(lambdas)):
    plt.plot(x_plot, predict(feature_mapping(x_plot,degree),w_opts_reg[i]), labe
l='$ \lambda=%s$' %str(lambdas[i]))

plt.title(r'$\bf{Figure\ 4.}$ Regularized Polynomial Regression')
plt.xlabel('Bitcoin')
plt.ylabel('Ethereum')
plt.ylim(0,1)
plt.xlim(0,1)
plt.legend()

#plot the training and validation errors for the different values of lambda
ax=fig.add_subplot(122)
```

```
plt.plot(lambdas,training_errors_reg,color='black',label='Training error')
plt.plot(lambdas,validation_errors_reg,color='red',label='Validation error')

plt.title(r'$\bf{Figure\ 5.}$ Training and validation error for different lambda
s')
plt.xlabel('Lambda')
plt.ylabel('Empirical error')
plt.xticks(lambdas)
plt.legend()
plt.show()
```

	Lambdas	Training errors	Validation errors
0	0.00	0.002970	1.346183
1	0.01	0.006321	0.021683
2	0.50	0.033803	0.056465
3	1.00	0.048999	0.076654
4	2.00	0.068277	0.101182
5	5.00	0.092993	0.131519

Figure 4. Regularized Polynomial Regression

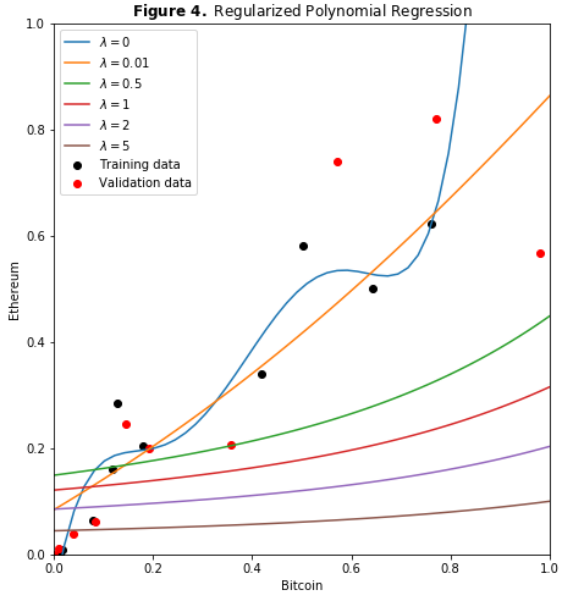


Figure 5. Training and validation error for different lambdas