

# MA 506 Probability and Statistical Inference

## Lecture 24: Regularized Classification

```
In [45]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

### Getting the data

```
In [6]: bc = load_breast_cancer()
#print(bc.DESCR)
```

```
In [7]: X = bc.data
y = bc.target
```

```
In [47]: #X[:,0]
```

### Standardizing the data

```
In [49]: scaler = StandardScaler()
X = scaler.fit_transform(X)
```

### Dividing data into training and testing set

```
In [50]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

```
In [51]: print(len(y_train), len(y_test))
```

455 114

## Dividing training set further into train and validation sets

```
In [52]: X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.2)
```

```
In [53]: print(len(y_tr), len(y_val))
```

```
318 137
```

## Getting logistic regression functions

```
In [62]: def h(X,beta):  
    ## returns the value of the sigmoid function  
    #print(X.dot(beta))  
    ypred = 1/(1 + np.exp(-1*X.dot(beta)))  
    return ypred  
  
def model_cost(X,y,beta):  
    ## computes the value of the model fitting cost  
    cost = 0  
    for j in range(X.shape[0]):  
        term1 = y[j]*np.log(h(X[j,:],beta))[0]  
        term2 = (1 - y[j])*np.log(1 - h(X[j,:],beta))[0]  
        cost = cost - (term1+term2)  
    return cost
```

## Avoiding overfitting

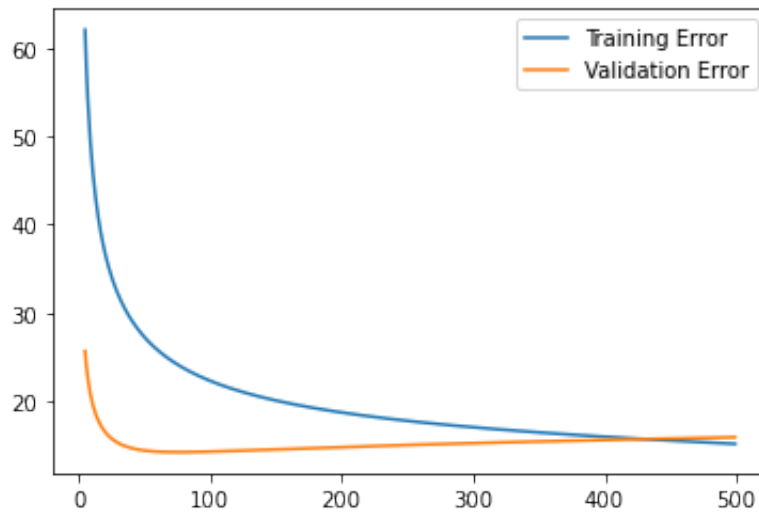
1. Early stopping

```
In [84]: def gradient_descent_validation(Xtr,ytr,Xval,yval,beta0,alpha,iteration)
        ## Implemented iteration for the gradient descent algorithm
        ypred = h(Xtr,beta0)
        beta = [beta0]
        train_cost = [model_cost(Xtr,ytr,beta0)]
        #print(model_cost(Xtr,ytr,beta0))
        #return
        val_cost = [model_cost(Xval,yval,beta0)]
        for j in range(iteration):
            grad_l = -1*Xtr.T.dot(ytr.reshape(-1,1) - ypred)
            beta_new = beta[-1] - alpha * grad_l
            ypred = h(Xtr,beta_new)
            beta.append(beta_new)
            print(f'At Iter {j}: Train cost:{model_cost(Xtr,ytr,beta_new)}')
            train_cost.append(model_cost(Xtr,ytr,beta_new))
            val_cost.append(model_cost(Xval,yval,beta_new))
        return [beta,np.array(train_cost),np.array(val_cost)]
```

```
In [88]: %%time
        beta = 0.1*np.ones([X.shape[1]+1,1])
        alpha = 0.001
        iteration = 500
        XX_tr = np.concatenate((np.ones([len(X_tr),1]),X_tr),axis = 1)
        XX_val = np.concatenate((np.ones([len(X_val),1]),X_val),axis = 1)
        beta,train_cost,val_cost = gradient_descent_validation(XX_tr,y_tr,XX_val,
                                                                beta,alpha,iteration)
```

```
At Iter 0: Train cost:129.06892653573428;Val cost: 55.419600182154205
At Iter 1: Train cost:89.89967638977046;Val cost: 37.49105914416196
At Iter 2: Train cost:75.96044523991692;Val cost: 31.428368802032306
At Iter 3: Train cost:67.73027447845591;Val cost: 28.00200609501544
At Iter 4: Train cost:62.03506424728679;Val cost: 25.710219067411185
At Iter 5: Train cost:57.76696772002614;Val cost: 24.03764588609951
At Iter 6: Train cost:54.4095157517202;Val cost: 22.749845837302935
At Iter 7: Train cost:51.679214366036824;Val cost: 21.721638663588227
At Iter 8: Train cost:49.403501016448914;Val cost: 20.878808242615
At Iter 9: Train cost:47.469752246164624;Val cost: 20.17398483817893
At Iter 10: Train cost:45.800710938702736;Val cost: 19.57523295892134
At Iter 11: Train cost:44.34134931261389;Val cost: 19.060093992744584
At Iter 12: Train cost:43.05128726505472;Val cost: 18.61223412865172
At Iter 13: Train cost:41.90015845651864;Val cost: 18.219443531750173
At Iter 14: Train cost:40.86464970768929;Val cost: 17.87238387609184
At Iter 15: Train cost:39.926541212102904;Val cost: 17.56377304459911
7
At Iter 16: Train cost:39.07137067797086;Val cost: 17.28783673550352
At Iter 17: Train cost:38.287499842431345;Val cost: 17.03992918630998
7
```

```
In [89]: index = list(range(5,500))  
plt.plot(index,train_cost[index],label = 'Training Error')  
plt.plot(index,val_cost[index],label = 'Validation Error')  
plt.legend();
```



## 2. Stochastic Gradient Descent

```
In [ ]:
```