Práctica 5: Regresión lineal regularizada: sesgo y varianza

Ana Martín Sánchez, Nicolás Pastore Burgos

21/09/2021

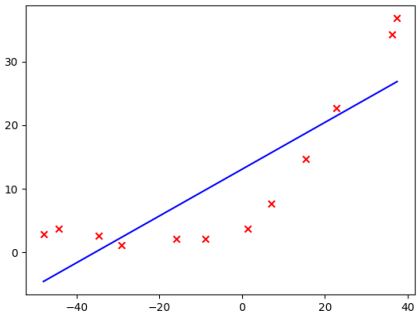
1 Descripción de la práctica

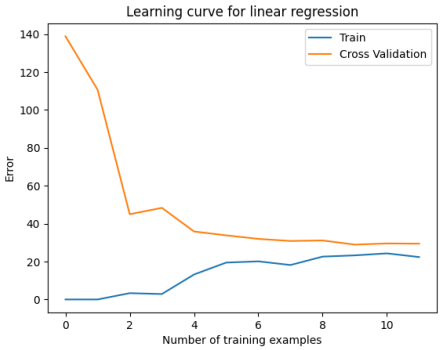
El objetivo de esta práctica era comprobar los efectos del sesgo y de la varianza. Para ello, se proponía aprender una hipótesis sesgada (que no fuese capaz siquiera de clasificar correctamente los ejemplos de entrenamiento), y posteriormente aplicar de nuevo la regresión lineal, para sobre-ajustar los datos de entrenamiento a un polinomio de mayor grado.

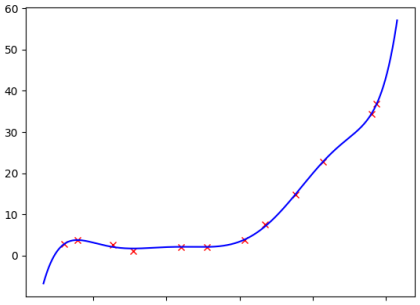
Los datos que se proporcionan en este caso se corresponden con los datos históricos del agua derramada de una presa, dependiendo de los cambios en el nivel del agua.

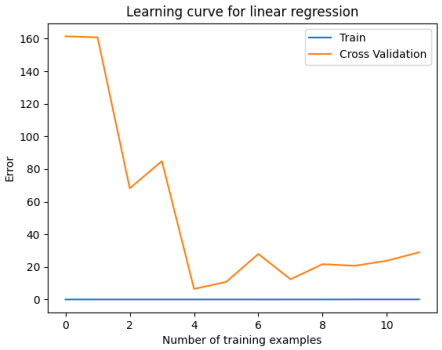
2 Solución propuesta

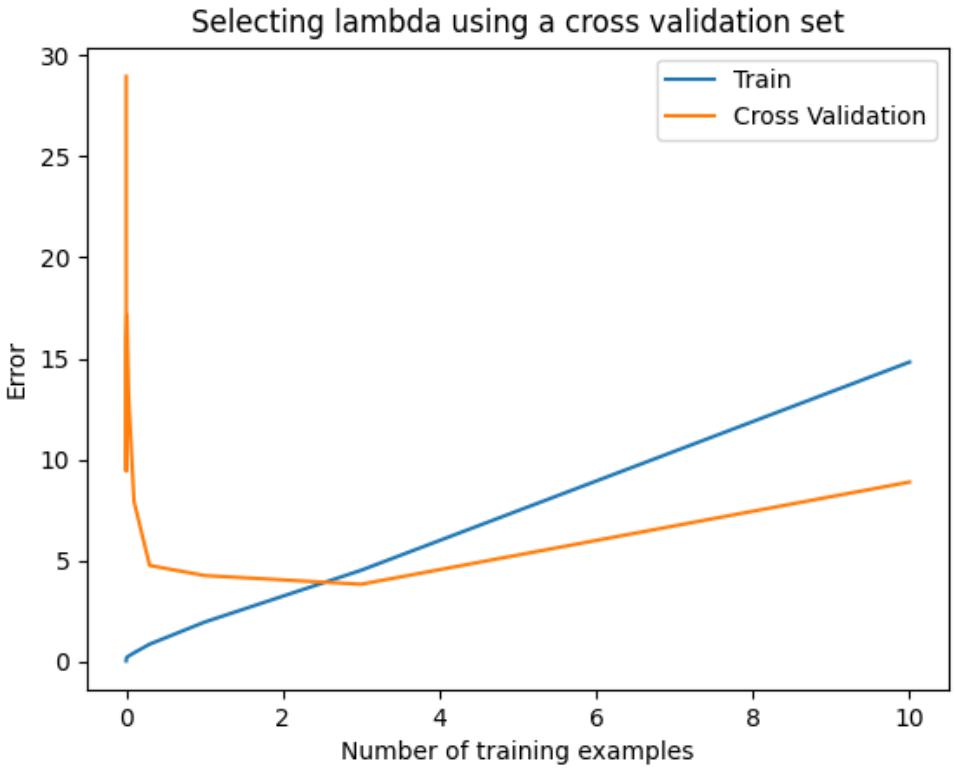
2.1 Resultados obtenidos











2.2 Implementación

```
1000 import numpy as np
1001 import matplotlib.pyplot as plt
1002
1003 from scipy.io import loadmat
1004
```

```
1005 from scipy.optimize import minimize
1006
1007 def costeRegul(thetas, x, y, reg):
1008     m = np.shape(x)[0]
1009     return (np.sum((np.dot(x,thetas) - y.T) ** 2)) / (2 * m) + (reg / (2 *
     m)) * np.sum(thetas[1:] ** 2)
1010
1011 def gradiente(thetas, x, y):
1012     return (np.dot((np.dot(x, thetas) - y.T), x)) / x.shape[0]
1013
1014 def gradienteRegul(thetas, x, y, reg):
1015     grad = gradiente(thetas, x, y)
1016
1017     res = grad + thetas * (reg / x.shape[0])
1018     res[0] = grad[0]
1019     return res
1020
1021 def desc_grad(thetas, x, y, reg):
1022     return costeRegul(thetas, x, y, reg), gradienteRegul(thetas, x, y, reg)
1023
1024 def parte1(x, xNew, y, n, reg):
1025     thetas = np.ones(n+1)
1026
1027     grad = gradienteRegul(thetas, xNew, y, reg)
1028     print("Gradient of thetas [1,1]: " + str(grad))
1029
1030     cost = costeRegul(thetas, xNew, y, reg)
1031     print("Cost of thetas [1,1]: " + str(cost))
1032
1033     res = minimize(desc_grad, x0=thetas, args=(xNew, y, reg), jac=True,
     method='TNC')
1034
1035     plt.figure()
1036     minX = np.min(x)
1037     maxX = np.max(x)
1038     minY = res.x[0] + res.x[1]*minX
1039     maxY = res.x[0] + res.x[1]*maxX
1040     plt.scatter(x, y, color='red', marker='x')
1041     plt.plot([minX, maxX], [minY, maxY], color='blue')
1042     plt.show()
1043
1044 """
1045     ==========================================================
1046     ==========================================================
1047     ==========================================================
1048     ==========================================================
1049 """
1050
1051 def parte2(x, xVal, y, yVal, reg):
1052     m = np.shape(x)[0]
1053     XvalNew = np.hstack([np.ones([np.shape(xVal)[0],1]),xVal])
1054
1055     train = np.zeros(m)
```

```
    val = np.zeros(m)

    for i in np.arange(1, m+1):
        xEval = x[: i]
        yEval = y[: i]

        thetas = np.zeros(np.shape(x)[1])
        res = minimize(desc_grad, x0=thetas, args=(xEval, yEval, reg), jac=True, method='TNC')
        train[i-1] = costeRegul(res.x, xEval, yEval, reg)
        val[i-1] = costeRegul(res.x, XvalNew, yVal, reg)

    plt.figure()
    plt.title('Learning curve for linear regression')
    plt.plot(np.linspace(0, m-1, m, dtype=int), train, label='Train')
    plt.plot(np.linspace(0, m-1, m, dtype=int), val, label='Cross Validation')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.legend()
    plt.show()

"""
    ==========================================================
    ==========================================================
    ==========================================================
    ==========================================================
"""

def createPoliX(x, p):
    xR = np.ravel(x)
    return np.array([(xR * (xR ** i)) for i in np.arange(p)]).T

def normalizaMat(mat):
    mu = np.array(np.mean(mat, axis=0))
    sigma = np.array(np.std(mat, axis=0))

    matNorm = (mat - mu) / sigma

    return matNorm, mu, sigma

def parte3(x, p, y, reg):
    matPoli = createPoliX(x, p)

    matNorm, mu, sigma = normalizaMat(matPoli)

    matNorm = np.hstack([np.ones([np.shape(matNorm)[0],1]),matNorm])

    thetas = np.zeros(matNorm.shape[1])

    res = minimize(desc_grad, x0=thetas, args=(matNorm, y, reg), jac=True, method='TNC')

```

```
1106      plt.plot(x, y, "x", color='red')
1107
1108      margin = 5.65
1109      lineX = np.arange(np.min(x) - margin,np.max(x) + margin,0.05)
1110      valsX = (createPoliX(lineX, p)-mu) / sigma
1111      lineY = np.dot(np.hstack([np.ones([len(valsX),1]),valsX]), res.x)
1112      plt.plot(lineX, lineY, '-', c = 'blue')
1113      plt.show()
1114
1115  def parte3_2(x, p, xVal, y, yVal, reg):
1116      matNorm, mu, sigma = normalizaMat(createPoliX(x, p))
1117      matNorm = np.hstack([np.ones([np.shape(matNorm)[0],1]),matNorm])
1118
1119      matXValNorm = (createPoliX(xVal, p) - mu) / sigma
1120
1121      parte2(matNorm, matXValNorm, y, yVal, reg)
1122
1123  """
1124      ==========================================================
1125      ==========================================================
1126      ==========================================================
1127      ==========================================================
1128  """
1129
1130  def parte4(x, p, xVal, y, yVal):
1131      lambdas = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])
1132
1133      lambdasSize = lambdas.shape[0]
1134
1135      matNorm, mu, sigma = normalizaMat(createPoliX(x,p))
1136      matNorm = np.hstack([np.ones([np.shape(matNorm)[0],1]),matNorm])
1137
1138      matValNorm = (createPoliX(xVal,p) - mu) / sigma
1139      matValNorm = np.hstack([np.ones([np.shape(matValNorm)[0],1]),matValNorm
          ])
1140
1141      train = np.zeros(lambdasSize)
1142      val = np.zeros(lambdasSize)
1143
1144      for i in np.arange(lambdasSize):
1145          thetas = np.zeros(np.shape(matNorm)[1])
1146          res = minimize(desc_grad, thetas, args=(matNorm, y, lambdas[i]),
          jac= True, method='TNC')
1147          train[i] = costeRegul(res.x, matNorm, y, 0)
1148          val[i] = costeRegul(res.x, matValNorm, yVal, 0)
1149
1150      plt.title('Selecting lambda using a cross validation set')
1151      plt.plot(lambdas,train,label="Train")
1152      plt.plot(lambdas,val,label="Cross Validation")
1153      plt.xlabel('Number of training examples')
1154      plt.ylabel('Error')
1155      plt.legend()
1156      plt.show()
```

```
1157
1158 def parte4_2(x, p, xTest, y, yTest, reg):
1159     matNorm, mu, sigma = normalizaMat(createPoliX(x,p))
1160     matNorm = np.hstack([np.ones([np.shape(matNorm)[0],1]),matNorm])
1161
1162     thetas = np.ones(matNorm.shape[1])
1163
1164     res = minimize(desc_grad, thetas, args=(matNorm, y, reg), jac=True,
        method='TNC')
1165
1166     matTest = (createPoliX(xTest,p) - mu) / sigma
1167     matTest = np.hstack([np.ones([np.shape(matTest)[0],1]),matTest])
1168
1169     error = costeRegul(res.x, matTest, yTest, 0)
1170     print("Error para lambda = " + str(reg) + ": " + str(error))
1171
1172 def main():
1173     data = loadmat("Data/ex5data1.mat")
1174
1175     x = data['X']
1176     y = data['y']
1177     yR = np.ravel(y)
1178
1179     xVal = data['Xval']
1180     yVal = data['yval']
1181     yValR = np.ravel(yVal)
1182
1183     xTest = data['Xtest']
1184     yTest = data['ytest']
1185
1186     m = np.shape(x)[0]
1187     n = np.shape(x)[1]
1188
1189     xNew = np.hstack([np.ones([m, 1]), x])
1190
1191     p = 8
1192
1193     reg = 1
1194     parte1(x, xNew, yR, n, reg)
1195     parte2(xNew, xVal, yR, yValR, reg)
1196
1197     reg = 0
1198     parte3(x, p, yR, reg)
1199     parte3_2(x, p, xVal, yR, yValR, reg)
1200
1201     reg = 3
1202     parte4(x, p, xVal, yR, yVal)
1203     parte4_2(x, p, xTest, yR, yTest, reg)
1204
1205 if __name__ == "__main__":
1206     main()
```

main.py