Aprendizaje Automático y Minería de Datos: Proyecto Final

Ana Martín Sánchez, Nicolás Pastore Burgos
Repositorio en GitHub

21/09/2021

1 Propuesta de proyecto

Nuestro proyecto consiste en clasificar setas como comestibles o venenosas, dependiendo de un total de 20 atributos, como los siguientes:

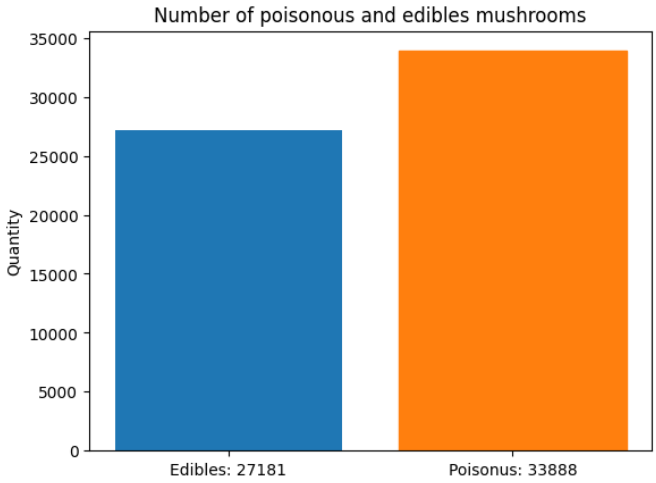
- Diámetro del sombrero: un float que representa el diámetro, en cm.
- Forma del sombrero: un char que representa una de las posibles formas.
- Superficie del sombrero: un char que representa el adjetivo que mejor describe la superficie del sombrero.

Para poder analizar los datos, desplegamos algunos de estos atributos (los que eran de tipo enumerado) a matrices de booleanos. Después de esta operación, cada entrada de la base de datos tenía 122 atributos.

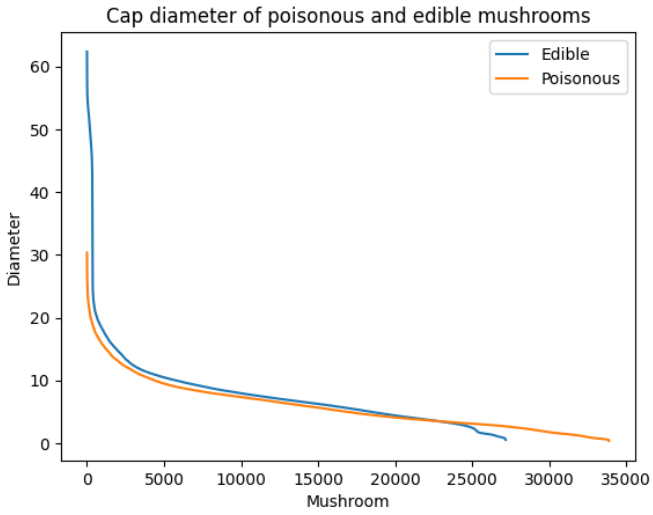
La base de datos tiene un total de 61069 entradas. Por este motivo, no hemos utilizado todos los datos para el proyecto; para las pruebas, de manera general, hemos escogido un 20% de los datos para entrenar los sistemas, otro 20% para validarlos y otro 20% para hacer la prueba final.

La base de datos original, extraída de la plataforma Kaggle, tenía los datos organizados de manera que escoger una muestra en el orden establecido no era útil. Por esta razón, decidimos utilizar los datos según estaban organizados en este repositorio, ya que estaban mezclados.

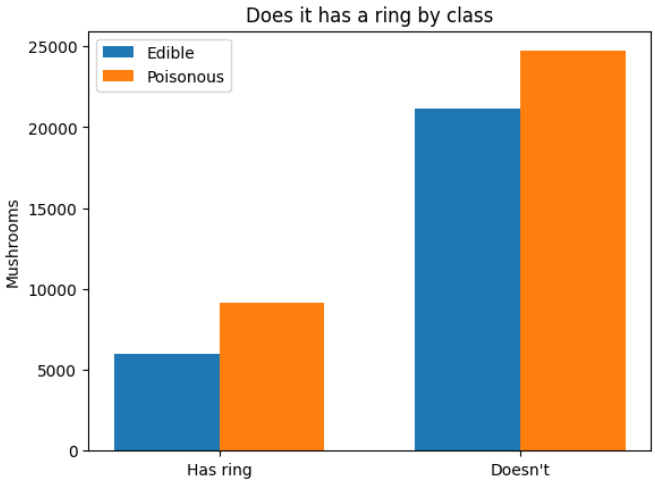
Para entender mejor los datos, presentamos una serie de gráficos que permiten hacer comparaciones entre las setas comestibles y las venenosas:



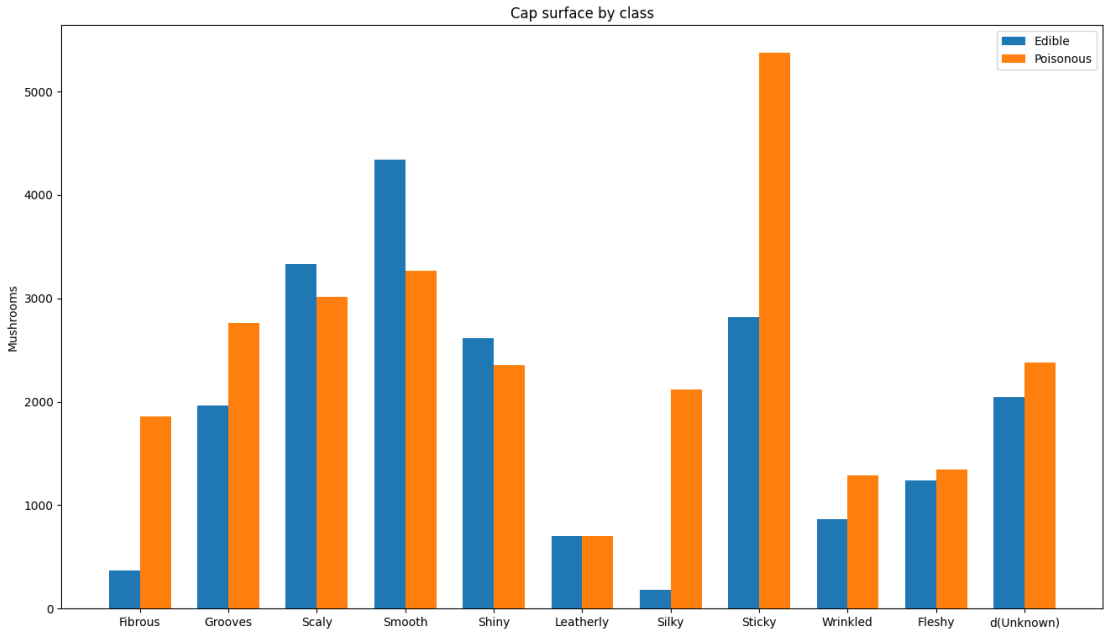
Como se puede observar, la cantidad de muestras de setas venenosas supera en número a la de setas comestibles, en más de 6000 entradas.



Nos pareció interesante mostrar cómo, cuando el diámetro del sombrero de las setas es muy grande, es muy probable que las setas sean comestibles. Sin embargo, en tamaños intermedios, resulta más difícil diferenciar las setas comestibles de las venenosas.



En este caso, vemos que la mayoría de las setas no tienen anillo. Hemos escogido esta gráfica como muestra de algunos de los valores binarios que se van a analizar más adelante.



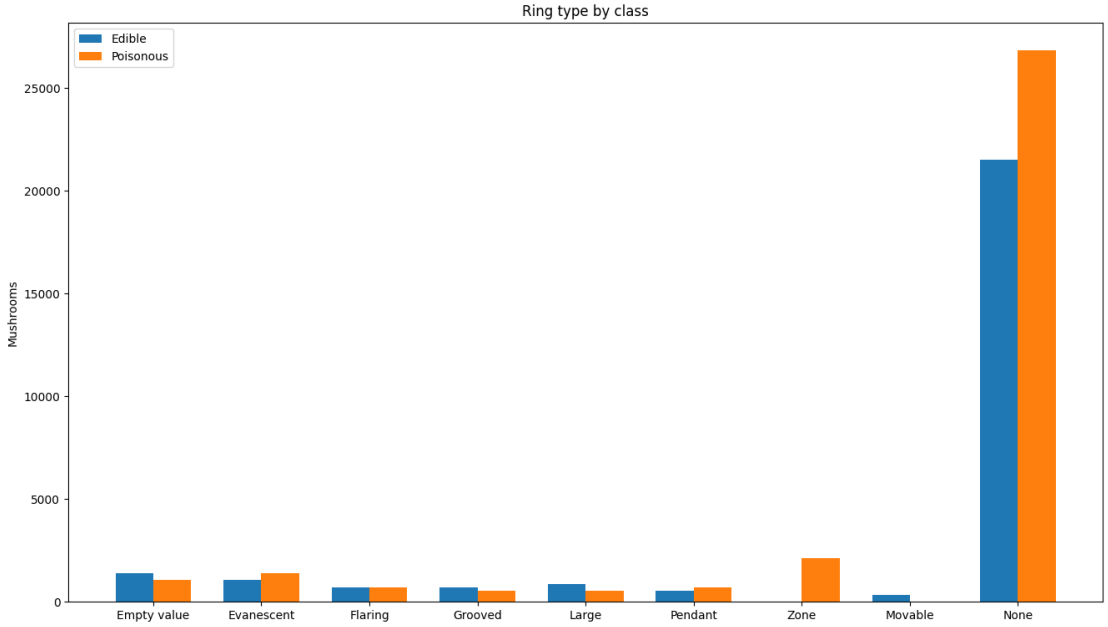
Por último, se muestran una serie de adjetivos que describen más fiablemente el tipo de superficie del sombrero, y cuáles son venenosas y comestibles. Como se puede ver, y como comentamos a continuación, muchas de las entradas tenían un valor "d" para esta característica (un valor desconocido).

1.1 Problemas encontrados

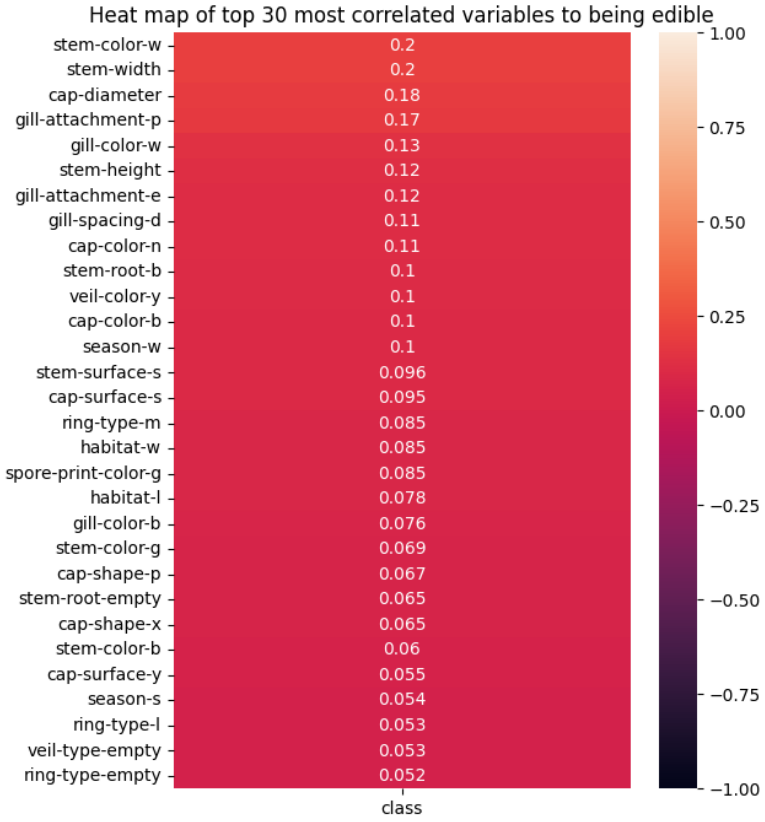
Al analizar los datos, nos dimos cuenta de que la base de datos nos supondría algunos problemas para realizar el proyecto:

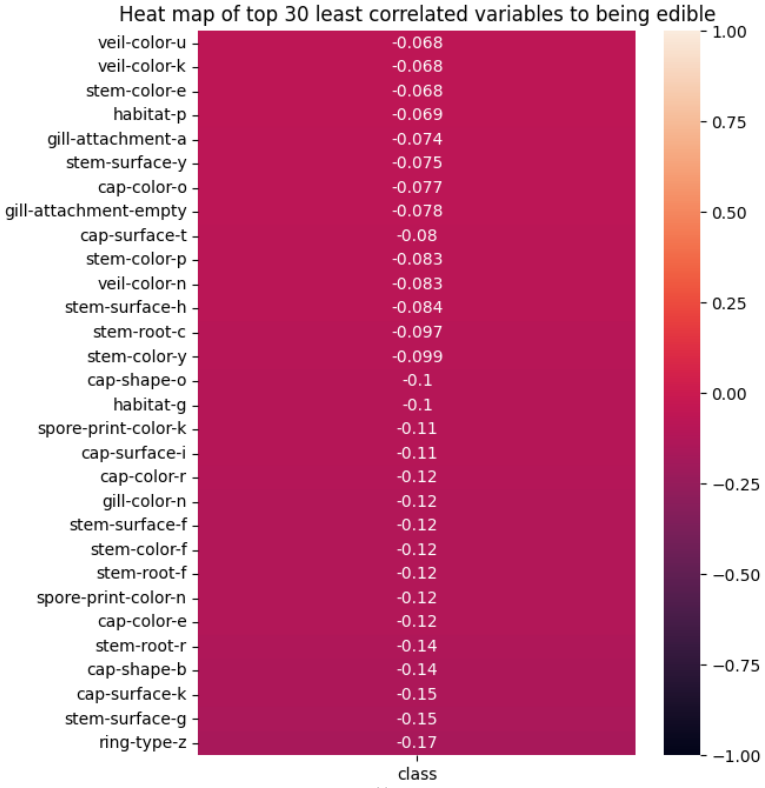
En primer lugar, hay columnas en las que todos los valores son 0. También se dio que, en la gran mayoría de filas, alguna de las columnas no tenía un valor.

Por otra parte, algunos de los datos eran contradictorios. En concreto, algunas de las entradas tenían marcado como "t" la columna de has-ring (lo que significa que, efectivamente, tienen un anillo); pero, posteriormente, en la columna de ring-type, tenían marcado "f" (que se corresponde con el tipo "ninguno"). Esto no supone un problema a la hora de implementar los sistemas de aprendizaje automático, pero sí son un problema desde el punto de vista semántico, y nos hacen dudar de la validez de los datos.



Por último, encontramos un problema con la correlación entre los datos y la salida. Como se puede observar en las siguientes gráficas, ninguna de las entradas tiene una correlación que supere el 0.2, ni en el eje positivo ni en el negativo.


Heat map of top 30 most correlated variables to being edible
stem-color-w 0.2
stem-width 0.2
cap-diameter 0.18
gill-attachment-p 0.17
gill-color-w 0.13
stem-height 0.12
gill-attachment-e 0.12
gill-spacing-d 0.11
cap-color-n 0.11
stem-root-b 0.1
veil-color-y 0.1
cap-color-b 0.1
season-w 0.1
stem-surface-s 0.096
cap-surface-s 0.095
ring-type-m 0.085
habitat-w 0.085
spore-print-color-g 0.085
habitat-l 0.078
gill-color-b 0.076
stem-color-g 0.069
cap-shape-p 0.067
stem-root-empty 0.065
cap-shape-x 0.065
stem-color-b 0.06
cap-surface-y 0.055
season-s 0.054
ring-type-l 0.053
veil-type-empty 0.053
ring-type-empty 0.052
class
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Heat map of top 30 least correlated variables to being edible
veil-color-u -0.068
veil-color-k -0.068
stem-color-e -0.068
habitat-p -0.069
gill-attachment-a -0.074
stem-surface-y -0.075
cap-color-o -0.077
gill-attachment-empty -0.078
cap-surface-t -0.08
stem-color-p -0.083
veil-color-n -0.083
stem-surface-h -0.084
stem-root-c -0.097
stem-color-y -0.099
cap-shape-o -0.1
habitat-g -0.1
spore-print-color-k -0.11
cap-surface-i -0.11
cap-color-r -0.12
gill-color-n -0.12
stem-surface-f -0.12
stem-color-f -0.12
stem-root-f -0.12
spore-print-color-n -0.12
cap-color-e -0.12
stem-root-r -0.14
cap-shape-b -0.14
cap-surface-k -0.15
stem-surface-g -0.15
ring-type-z -0.17
class
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

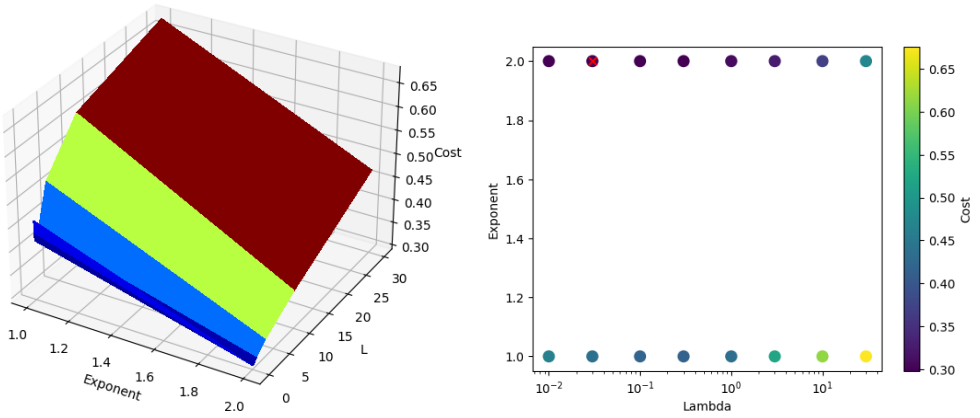
2 Resultados obtenidos

Para obtener los mejores resultados posibles, hemos utilizado varios métodos de los estudiados en clase, a fin de poder compararlos.

2.1 Regresión Logística

La regresión logística se puede entender como un caso especial de la regresión lineal, en la que la variable dependiente puede tomar dos valores: 0 o 1. Se emplea para calcular probabilidades (ya que los valores obtenidos estarán entre 0 y 1), o para clasificar eventos en dos categorías; en este caso, se puede utilizar para calcular la probabilidad de que una seta sea comestible (o, dicho de otra manera, para clasificar una seta como comestible o venenosa).

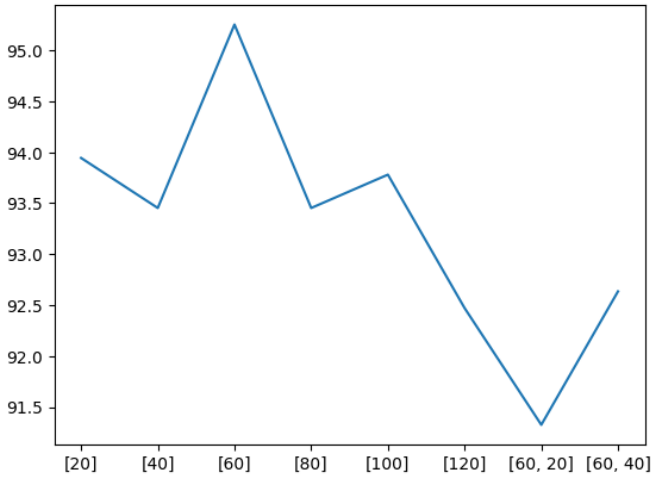
Para utilizar este método, implementamos la función de gradiente, de coste y la función sigmoide como hemos visto en clase, y obtuvimos una tasa de aciertos del 97.14262%, con un coste de 0.12467. Para llegar a estos resultados, utilizamos un valor para lambda = 3.0 y un exponente = 2.



2.2 Redes Neuronales

Se entiende como red neuronal a un conjunto de capas ocultas, una capa de entrada y una capa de salida que intentan imitar el funcionamiento del cerebro humano. Cada nodo de las capas (el que equivaldría a una "neurona artificial"), se conecta a otros, y tiene un peso y umbral asociados. Si el valor de un nodo supera su umbral, la "neurona" se activa y envía los datos a la siguiente capa de la red.

Tras probar con varias configuraciones para la red neuronal, llegamos a un porcentaje de aciertos del 99.828%, con un coste de 0.0025. Para ello, utilizamos una red neuronal con dos capas ocultas (la primera, con 60 nodos; y la segunda, con 40), y una constante de regularización de 0.3. El algoritmo dio 1000 vueltas para llegar a este resultado.



2.3 SVM

Una Máquina de Vectores de Soporte (o SVM, por sus siglas en inglés), es un conjunto de algoritmos relacionados con problmas de clasificación y regresión. Una SVM intenta construir un hiperplano de un espacio N-dimensional que consiga clasificar los datos correctamente (donde N es el número de características que se proporcionan).

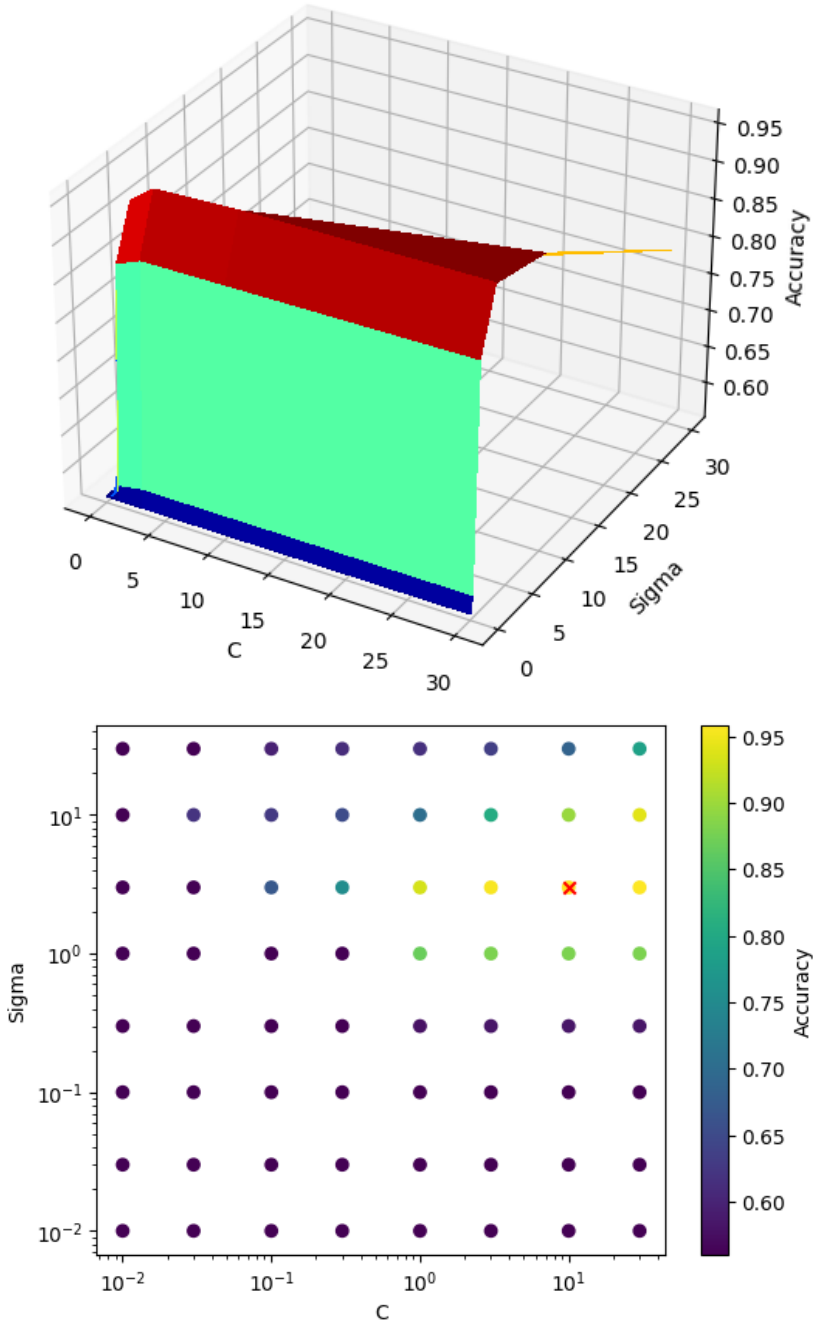
Entre las pruebas que realizamos, comparamos las que nos dieron los siguientes resultados:

Resultados obtenidos al aplicar SVM a nuestra base de datos.						
	C	Sigma	Porcentaje de datos usados (Training - Validation - Test)	Mejor probabilidad de acierto	Porcentaje de aciertos (%)	Tiempo de análisis (s)
1	3.0	3.0	0.20 - 0.20 - 0.20	0.999	99.984	5572
2	3.0	3.0	0.20 - 0.20 - 0.20	0.999	99.984	3856
3	10.0	3.0	0.05 - 0.05 - 0.05	0.996	99.705	104
4	10.0	3.0	0.01 - 0.01 - 0.01	0.959	95.908	1.95

Entre las dos primeras pruebas que se muestran, es notable la diferencia de tiempos. Esto se debe a que, después de varias pruebas, decidimos utilizar hebras para agilizar este proceso.

Entre los siguientes dos sets, el tiempo también se reduce dráticamente, pero por otro motivo: en estos casos, escogimos muestras más pequeñas para hacer las pruebas. Es interesante observar que, aunque el proceso es más rápido, el porcentaje de aciertos también decae.

Nuestro mejor resultado se corresponde, entonces, con la segunda entrada de la tabla: un 99.984% de aciertos con un valor de C = 3.0 y un valor de sigma = 3.0, y reduciendo en casi 30 minutos el tiempo necesario para analizar los datos.

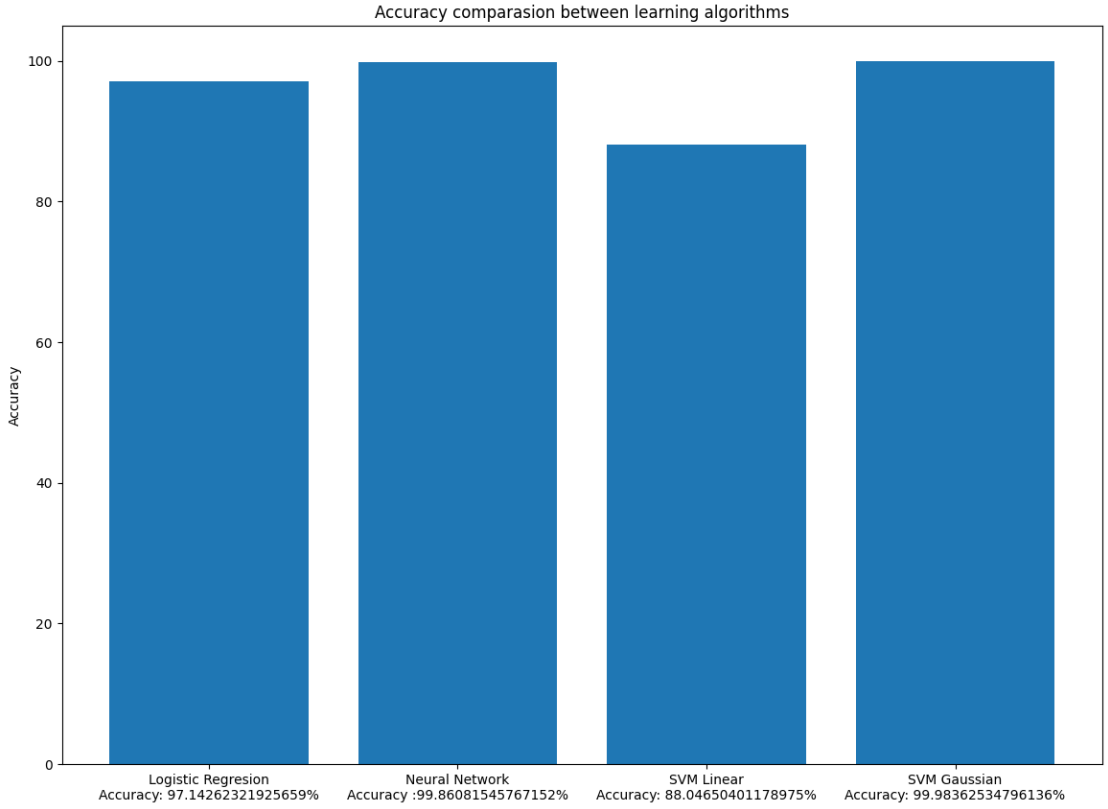

0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Accuracy
0
5
10
15
20
25
30
C
0
5
10
15
20
25
30
Sigma
10^{1}
10^{0}
10^{-1}
10^{-2}
Sigma
10^{-2}
10^{-1}
10^{0}
10^{1}
C
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Accuracy

3 Conclusiones

Para calcular los algoritmos, utilizamos en todos los casos unos datasets con las siguientes características:

- Tamaño de la muestra de entrenamiento: 12213 entradas
- Tamaño de la muestra de validación: 12213 entradas
- Tamaño de la muestra de testeo: 12213 entradas
- En todos los casos, las muestras son diferentes entre sí.

Tras realizar todas las pruebas, obtuvimos los siguientes resultados:



3.1 Implementación

Para conseguir los resultados anteriores, implementamos las siguientes funciones, repartidas en varios archivos:

```
1000 import numpy as np
1001
1002 import sklearn.preprocessing as sk
1003 import scipy.optimize as opt
1004
1005 import matplotlib.pyplot as plt
1006 from matplotlib import cm
1007 from mpl_toolkits.mplot3d import Axes3D
1008
1009 import time
1010
1011 from threadRetVal import ThreadWithReturnValue
1012
1013 def sigmoide(z):
1014     return (1 / (1 + np.exp(-z)))
1015
1016 def coste(thetas, x, y):
1017     h = sigmoide(np.dot(x, thetas))
1018     return -1/len(x) * (np.dot(np.log(h), y) + np.dot(np.log(1-h), 1-y))
1019
1020 def costeReg(thetas, x, y, l):
1021     h = sigmoide(np.dot(x, thetas))
1022     return -((np.dot(np.log(h), y) + np.dot(np.log(1 - h), 1 - y)) / len(x)
     ) + (l/(2*len(x))) * l * np.sum(thetas[1:] ** 2)
1023
1024 def gradienteReg(thetas, x, y, l):
1025     h = sigmoide(np.dot(x, thetas))
1026     return np.dot(x.T, h-y) / len(y) + (thetas * l) / len(y)
1027
1028 def threadMethod(xPolTrain, xPolVal, yTrain, yVal, n, exp, l):
1029     print('Testing for exp: ' + str(exp) + ' and lambda: ' + str(l))
1030     thetas = np.zeros(n)
1031     thetas = opt.fmin_tnc(func=costeReg, x0=thetas, disp=False, fprime=
     gradienteReg, args=(xPolTrain,yTrain,l))[0]
1032     cost = coste(thetas,xPolVal,yVal)
1033     print('Completed test for exp: ' + str(exp) + ' and lambda: ' + str(l))
1034     return thetas, cost
1035
1036 def evalLogisticReg(xTrain, xVal, yTrain, yVal):
1037     ls = np.array([0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0, 30.0])
1038     exps = np.array([1, 2]) # 3 asks for too much memory
1039
1040     numLs = ls.shape[0]
1041     numExps = exps.shape[0]
1042
1043     pol = np.empty(numExps, dtype=object)
1044
1045     for i in np.arange(numExps):
1046         pol[i] = sk.PolynomialFeatures(exps[i])
```

```
1047
1048        startTime = time.time()
1049
1050        resCost = np.zeros(numExps * numLs).reshape(numExps, numLs)
1051        resThet = np.empty_like(resCost, dtype=object)
1052        threads = np.empty_like(resCost, dtype=object) # can't be used, for
            lack of storage
1053
1054        for i in np.arange(numExps):
1055            xPolTrain = pol[i].fit_transform(xTrain)
1056            xPolVal = pol[i].fit_transform(xVal)
1057            n = np.shape(xPolTrain)[1]
1058            for j in np.arange(numLs):
1059                threads[i,j] = ThreadWithReturnValue(target=threadMethod, args
                =(xPolTrain, xPolVal, yTrain, yVal, n, exps[i], ls[j],))
1060                threads[i,j].start()
1061
1062        for i in np.arange(numExps):
1063            for j in np.arange(numLs):
1064                resThet[i,j], resCost[i,j] = threads[i,j].join()
1065
1066        bestCost = np.min(resCost)
1067        w = np.where(resCost == bestCost)
1068        bestExpIndex = w[0][0]
1069        bestLIndex = w[1][0]
1070        bestTheta = resThet[bestExpIndex, bestLIndex]
1071
1072        print()
1073        print('Best cost: ' + str(bestCost))
1074        print('Best lambda: ' + str(ls[bestLIndex]))
1075        print('Best exponent: ' + str(exps[bestExpIndex]))
1076
1077        endTime = time.time()
1078        print('Seconds elapsed of test: ' + str(endTime - startTime))
1079        print()
1080
1081        fig = plt.figure()
1082
1083        expexp, ll = np.meshgrid(ls, exps)
1084
1085        ax = Axes3D(fig, auto_add_to_figure=False)
1086        ax.set_xlabel('Exponent')
1087        ax.set_ylabel('L')
1088        ax.set_zlabel('Cost')
1089
1090        fig.add_axes(ax)
1091
1092        ax.plot_surface(ll,expexp,resCost, cmap=cm.jet, linewidth=0,
            antialiased=False)
1093        plt.savefig('../Results/LogReg/LOG_1.png', bbox_inches='tight')
1094        plt.close()
1095
1096        plt.scatter(expexp, ll, s=80, c=resCost)
```

```
    plt.xscale('log')
    plt.xlabel('Lambda')
    plt.ylabel('Exponent')
    plt.clim(np.min(resCost), np.max(resCost))
    plt.colorbar().set_label('Cost')
    plt.scatter(ls[bestLIndex], exps[bestExpIndex], s=40, marker='x', color='r')
    plt.savefig('../Results/LogReg/LOG_2.png', bbox_inches='tight')
    plt.close()

    xPolVal = pol[bestExpIndex].fit_transform(xVal)

    res = sigmoide(np.dot(bestTheta, xPolVal.T))
    acertados = np.sum((res >= 0.5) == yVal)
    accuracy = acertados*100/np.shape(res)[0]
    print("Accuracy of train: " + str(accuracy) + "%")

    return bestTheta, pol[bestExpIndex], accuracy, bestCost, ls[bestLIndex], exps[bestExpIndex]

def getNumAcertadosLog(theta, x, y):
    res = sigmoide(np.dot(theta, x.T))
    return np.sum((res >= 0.5) == y)
```

src/evaluateLogistic.py

```
import numpy as np

from scipy.optimize import minimize

def sigmoide(z):
    return 1 / (1 + np.exp(-z)) + 1e-9

def forwardProp(x, num_capas, thetas):
    a = np.empty(num_capas + 1, dtype="object")
    a[0] = x
    for i in range(num_capas):
        aNew = np.hstack([np.ones([x.shape[0], 1]), a[i]])
        a[i] = aNew
        a[i+1] = sigmoide(np.dot(aNew,thetas[i].T))
    return a

def coste(x, y_ones, num_capas, thetas):
    res = forwardProp(x, num_capas, thetas)[num_capas]

    return np.sum((-(y_ones) * np.log(res)) - ((1 - y_ones) * np.log(1-res))) / x.shape[0]

def costeRegul(x, y_ones, num_capas, thetas, reg):
    cost = coste(x, y_ones, num_capas, thetas)

    val = 0

    for i in range(num_capas):
```

```
1027            val += np.sum(np.power(thetas[i][1:], 2))
1028
1029        regul = val * (reg / (2*x.shape[0]))
1030
1031        return cost + regul
1032
1033    def grad_Delta(m, Delta, theta, reg):
1034        gradient = Delta
1035
1036        col0 = gradient[0]
1037        gradient = gradient + (reg/m)*theta
1038        gradient[0] = col0
1039
1040        return gradient
1041
1042    def lineal_back_prop(x, y, thetas, reg):
1043        m = x.shape[0]
1044        Delta1 = np.zeros_like(thetas[0])
1045        Delta2 = np.zeros_like(thetas[1])
1046
1047        hThetaTot = forwardProp(x, thetas.shape[0], thetas)
1048
1049        dlts = np.empty_like(thetas)
1050        Deltas = np.empty_like(thetas)
1051
1052        dlts[-1] = (hThetaTot[-1].T - y).T
1053
1054        for t in range(m):
1055            a1t = hThetaTot[0][t, :] # (401,)
1056            a2t = hThetaTot[1][t, :] # (26,)
1057            ht = hThetaTot[2][t, :] # (10,)
1058            yt = np.reshape(y[t], (1,)) # (10,)
1059
1060            d3t = ht - yt # (10,)
1061            d2t = np.dot(thetas[1].T, d3t) * (a2t * (1 - a2t)) # (26,)
1062
1063            Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
1064            Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])
1065
1066        Delta1 = Delta1 / m
1067        Delta2 = Delta2 / m
1068
1069        gradient1 = grad_Delta(m, Delta1, thetas[0], reg)
1070        gradient2 = grad_Delta(m, Delta2, thetas[1], reg)
1071
1072        return np.append(gradient1, gradient2).reshape(-1)
1073
1074    def vect_back_prop(x, y, thetas, reg):
1075        m = x.shape[0]
1076
1077        hThetaTot = forwardProp(x, thetas.shape[0], thetas)
1078
1079        dlts = np.empty_like(thetas)
```

```
1080        Deltas = np.empty_like(thetas)
1081
1082        #dlts[-1] = (hThetaTot[-1].T - y).T
1083        dlts[-1] = hThetaTot[-1] - y
1084
1085        for i in range(1, thetas.shape[0]):
1086            a = hThetaTot[-(i+1)]
1087
1088            delta = np.dot(thetas[-i].T, dlts[-i].T).T
1089
1090            delta = delta * a * (1-a)
1091
1092            delta = delta[:,1:]
1093
1094            dlts[-(i+1)] = delta
1095
1096        res = []
1097
1098        for i in range(thetas.shape[0]):
1099            Deltas[i] = np.dot(dlts[i].T, hThetaTot[i]) / m
1100            Deltas[i] = np.append(Deltas[i][0], Deltas[i][1:] + (reg/m) *
        thetas[i][1:])
1101            res = np.append(res, Deltas[i]).reshape(-1)
1102
1103        return res
1104
1105    def backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, x, y, reg
        ):
1106        # backprop devuelve una tupla (coste, gradiente) con el coste y el
        gradiente de
1107        # una red neuronal de tres capas o mas, con num_entradas, num_ocultas
        nodos en las capas
1108        # ocultas y num_etiquetas nodos en la capa de salida. Si m es el numero
         de ejemplos
1109        # de entrenamiento, la dimension de 'X' es (m, num_entradas) y la de 'y
        '' es
1110        # (m, num_etiquetas)
1111
1112        if(num_ocultas.shape[0] + 2 < 3):
1113            print("ERROR: num_capas incorrect, must have an input, at least one
         hidden and an output layer")
1114            return (0,0)
1115
1116        # calculo de thetas
1117        thetas = np.empty(num_ocultas.shape[0] + 1, dtype='object')
1118        pointer = num_ocultas[0] * (num_entradas + 1)
1119
1120        thetas[0] = np.reshape(params_rn[: pointer] , (num_ocultas[0], (
        num_entradas + 1)))
1121
1122        for i in range(1, num_ocultas.shape[0]):
1123            thetas[i] = np.reshape(params_rn[pointer : pointer + num_ocultas[i]
         * (num_ocultas[i-1] + 1)], (num_ocultas[i], (num_ocultas[i-1] + 1)))
```

```
1124            pointer += num_ocultas[i] * (num_ocultas[i-1] + 1)
1125
1126        thetas[num_ocultas.shape[0]] = np.reshape(params_rn[pointer :] , (
            num_etiquetas, (num_ocultas[-1] + 1)))
1127
1128        #return costeRegul(x, y, thetas.shape[0], thetas, reg),
            lineal_back_prop(x, y, thetas, reg)
1129        return costeRegul(x, y, thetas.shape[0], thetas, reg), vect_back_prop(x
            , y, thetas, reg)
1130
1131    def getThetas(num_entradas, num_ocultas, num_etiquetas, out):
1132        thetas = np.empty(shape=[num_ocultas.shape[0] + 1], dtype='object')
1133        pointer = num_ocultas[0] * (num_entradas + 1)
1134
1135        thetas[0] = np.reshape(out.x[: pointer] , (num_ocultas[0], (
            num_entradas + 1)))
1136
1137        for i in range(1, num_ocultas.shape[0]):
1138            thetas[i] = np.reshape(out.x[pointer : pointer + num_ocultas[i] * (
            num_ocultas[i-1] + 1)], (num_ocultas[i], (num_ocultas[i-1] + 1)))
1139            pointer += num_ocultas[i] * (num_ocultas[i-1] + 1)
1140
1141        thetas[-1] = np.array(np.reshape(out.x[pointer :] , (num_etiquetas, (
            num_ocultas[-1] + 1))), dtype='float')
1142
1143        return thetas
1144
1145    def evalNN(num_entradas, num_ocultas, num_etiquetas, xTrain, xVal, yTrain,
        yVal, tagsTrain, tagsVal):
1146        eIni = np.sqrt(6) / np.sqrt(num_etiquetas + num_entradas) # = sqrt(6) /
             sqrt(Lin + Lout)
1147        regs = np.array([0.01, 0.03, 0.1, 0.3, 0.1, 0.3, 1.0, 3.0, 10.0, 30.0])
1148        laps = np.array([25, 50, 75, 100, 250, 500, 750, 1000])
1149
1150        numRegs = regs.shape[0]
1151        numLaps = laps.shape[0]
1152
1153        resCost = np.zeros(numRegs * numLaps).reshape(numRegs, numLaps)
1154        resThet = np.empty_like(resCost, dtype=object)
1155
1156        pesosSize = (num_entradas + 1) * num_ocultas[0] + (num_ocultas[-1] + 1)
             * num_etiquetas
1157
1158        for i in range(1, num_ocultas.shape[0]):
1159            pesosSize = pesosSize + ((num_ocultas[i-1] + 1) * num_ocultas[i])
1160
1161        randomTries = 1
1162
1163        bestCost = 10000
1164
1165        for i in np.arange(numRegs):
1166            for j in np.arange(numLaps):
1167                print(str(num_ocultas) + ' Testing for reg: ' + str(regs[i]) +
```

```
    ' and laps: ' + str(laps[j]))
1168            pesos = np.random.uniform(-eIni, eIni, pesosSize)
1169            # because its random, finds the best out of a number of trials
1170            for x in np.arange(randomTries):
1171                out = minimize(fun = backprop, x0= pesos,
1172                    args = (num_entradas, num_ocultas, num_etiquetas,
    xTrain, tagsTrain, regs[i]),
1173                    method='TNC', jac = True, options = {'maxiter': laps[j
    ]})
1174                thetas = getThetas(num_entradas, num_ocultas, num_etiquetas
    , out)
1175                cost = coste(xTrain, tagsTrain, thetas.shape[0], thetas)
1176                if(cost < bestCost):
1177                    bestCost = cost
1178                    bestTheta = thetas
1179                    bestRegIndex = i
1180                    bestLapsIndex = j
1181
1182    print()
1183
1184    print(str(num_ocultas) + " Best cost: " + str(bestCost))
1185    print(str(num_ocultas) + " Best Reg: " + str(regs[bestRegIndex]))
1186    print(str(num_ocultas) + " Best Laps: " + str(laps[bestLapsIndex]))
1187
1188    res = forwardProp(xVal, bestTheta.shape[0], bestTheta)[-1]
1189    maxIndices = np.argmax(res,axis=1)
1190    acertados = np.sum(maxIndices == yVal)
1191    accuracy = acertados*100/np.shape(res)[0]
1192    print(str(num_ocultas) + " Accuracy of train: " + str(accuracy) + "%")
1193    print()
1194
1195    return bestTheta, accuracy, bestCost, regs[bestRegIndex], laps[
    bestLapsIndex]
1196
1197 def getNumAcertadosNN(x, y, thetas):
1198    res = forwardProp(x, thetas.shape[0], thetas)[-1]
1199    maxIndices = np.argmax(res,axis=1)
1200    return np.sum(maxIndices == y)
```

src/evaluateNeuronal.py

```
1000 import numpy as np
1001
1002 import sklearn.svm as svm
1003
1004 import matplotlib.pyplot as plt
1005 from matplotlib import cm
1006 from mpl_toolkits.mplot3d import Axes3D
1007
1008 from sklearn.metrics import accuracy_score
1009
1010 import time
1011
1012 from threadRetVal import ThreadWithReturnValue
```

```
1013
1014 def threadMethod(x, xVal, y, yVal, c, sigma):
1015     print('Testing c ' + str(c) + ' sigma ' + str(sigma))
1016     s = svm.SVC(kernel='rbf', C=c, gamma= 1 / (2 * sigma ** 2))
1017     s.fit(x,y)
1018
1019     print('Completed c ' + str(c) + ' sigma ' + str(sigma))
1020     return s, accuracy_score(yVal, s.predict(xVal))
1021
1022 def evaluateSVM(x, xVal, y, yVal):
1023     startTime = time.time()
1024
1025     cs = np.array([0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30])
1026     sigmas = np.array([0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30])
1027
1028     numCs = cs.shape[0]
1029     numSigmas = cs.shape[0]
1030
1031     resAcc = np.zeros(numCs * numSigmas).reshape(numCs, numSigmas)
1032     resSVM = np.empty_like(resAcc, dtype=object)
1033     threads = np.empty_like(resAcc, dtype=object)
1034
1035     for i in np.arange(numCs):
1036         for j in np.arange(numSigmas):
1037             threads[i,j] = ThreadWithReturnValue(target=threadMethod, args
    =(x, xVal, y, yVal, cs[i], sigmas[j],))
1038             threads[i,j].start()
1039
1040     for i in np.arange(numCs):
1041         for j in np.arange(numSigmas):
1042             resSVM[i,j], resAcc[i,j] = threads[i,j].join()
1043
1044     bestAcc = np.max(resAcc)
1045     w = np.where(resAcc == bestAcc)
1046     bestC = cs[w[0][0]]
1047     bestSigma = sigmas[w[1][0]]
1048     bestSVM = resSVM[w[0][0],w[1][0]]
1049
1050     print()
1051     print("Best accuracy: " + str(bestAcc))
1052     print("Best C: " + str(bestC))
1053     print("Best Sigma: " + str(bestSigma))
1054
1055     endTime = time.time()
1056     print('Seconds elapsed of test: ' + str(endTime - startTime))
1057
1058     fig = plt.figure()
1059
1060     cc, ss = np.meshgrid(sigmas, cs)
1061
1062     ax = Axes3D(fig, auto_add_to_figure=False)
1063     ax.set_xlabel('C')
1064     ax.set_ylabel('Sigma')
```

```
1065      ax.set_zlabel('Accuracy')
1066
1067      fig.add_axes(ax)
1068
1069      ax.plot_surface(ss,cc,resAcc, cmap=cm.jet, linewidth=0, antialiased=
          False)
1070      plt.savefig('../Results/SVM/SVM_1.png', bbox_inches='tight')
1071      plt.close()
1072
1073      plt.scatter(ss, cc, c=resAcc)
1074      plt.xscale('log')
1075      plt.yscale('log')
1076      plt.xlabel('C')
1077      plt.ylabel('Sigma')
1078      plt.clim(np.min(resAcc), np.max(resAcc))
1079      plt.colorbar().set_label('Accuracy')
1080      plt.scatter(bestC, bestSigma, marker='x', color='r')
1081      plt.savefig('../Results/SVM/SVM_2.png', bbox_inches='tight')
1082      plt.close()
1083
1084      return bestSVM, bestAcc, bestC, bestSigma
1085
1086  def threadMethodLinear(x, xVal, y, yVal, c):
1087      print('Testing c ' + str(c))
1088      s = svm.SVC(kernel='linear', C=c)
1089      s.fit(x,y)
1090
1091      print('Completed c ' + str(c))
1092      return s, accuracy_score(yVal, s.predict(xVal))
1093
1094  def evaluateSVMLinear(x, xVal, y, yVal):
1095      startTime = time.time()
1096
1097      cs = np.array([0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30])
1098
1099      numCs = cs.shape[0]
1100
1101      resAcc = np.zeros(numCs)
1102      resSVM = np.empty_like(resAcc, dtype=object)
1103      threads = np.empty_like(resAcc, dtype=object)
1104
1105      for i in np.arange(numCs):
1106          threads[i] = ThreadWithReturnValue(target=threadMethodLinear, args
          =(x, xVal, y, yVal, cs[i],))
1107          threads[i].start()
1108
1109      for i in np.arange(numCs):
1110          resSVM[i], resAcc[i] = threads[i].join()
1111
1112      bestAcc = np.max(resAcc)
1113      w = np.where(resAcc == bestAcc)
1114      bestC = cs[w[0]]
1115      bestSVM = resSVM[w[0]][0]
```

```
1116
1117        print()
1118        print("Best accuracy: " + str(bestAcc))
1119        print("Best C: " + str(bestC))
1120
1121        endTime = time.time()
1122        print('Seconds elapsed of test: ' + str(endTime - startTime))
1123
1124        plt.plot(cs, resAcc)
1125        plt.xscale('log')
1126        plt.xlabel('C')
1127        plt.ylabel('Accuracy')
1128        plt.scatter(bestC, bestAcc, marker='x', color='r')
1129        plt.savefig('../Results/SVM/SVM_LIN.png', bbox_inches='tight')
1130        plt.close()
1131
1132        return bestSVM, bestAcc, bestC
```

src/evaluateSVM.py

```
1000 from enum import Enum
1001
1002 import numpy as np
1003 import matplotlib.pyplot as plt
1004 import seaborn as sns
1005
1006 #region Enums
1007 class CapShape(Enum):
1008         b = 0
1009         c = 1
1010         x = 2
1011         f = 3
1012         s = 4
1013         p = 5
1014         o = 6
1015
1016 class Surface(Enum):
1017         i = 1
1018         g = 2
1019         y = 3
1020         s = 4
1021         h = 5
1022         l = 6
1023         k = 7
1024         t = 8
1025         w = 9
1026         e = 10
1027         d = 11 # TO DO: wat
1028         f = 12
1029
1030 class Color(Enum):
1031         n = 1
1032         b = 2
1033         g = 3
```

```
        r = 4
        p = 5
        u = 6
        e = 7
        w = 8
        y = 9
        l = 10
        o = 11
        k = 12
        f = 13

class GillAttach(Enum):
        empty = 0
        a = 1
        x = 2
        d = 3
        e = 4
        s = 5
        p = 6
        f = 7

class GillSpacing(Enum):
        empty = 0
        c = 1
        d = 2
        f = 3

class StemRoot(Enum):
        empty = 0
        b = 1
        s = 2
        c = 3
        u = 4
        e = 5
        z = 6
        r = 7
        f = 8

class VeilType(Enum):
        empty = 0
        p = 1
        u = 2

class RingType(Enum):
        empty = 0
        c = 1
        e = 2
        r = 3
        g = 4
        l = 5
        p = 6
        s = 7
        z = 8
```

```
1087            y = 9
1088            m = 10
1089            f = 11
1090
1091    class Habitat(Enum):
1092            g = 0
1093            l = 1
1094            m = 2
1095            p = 3
1096            h = 4
1097            u = 5
1098            w = 6
1099            d = 7
1100
1101    class Season(Enum):
1102            s = 0
1103            u = 1
1104            a = 2
1105            w = 3
1106    #endregion
1107
1108    def prepareData(data):
1109        data['class'] = [int(i == 'e') for i  in data['class']]
1110
1111        for i in CapShape._member_names_:
1112            data['cap-shape-' + i] = np.where(data['cap-shape'] == i, 1, 0)
1113        data.pop('cap-shape')
1114
1115        for i in Surface._member_names_:
1116            data['cap-surface-' + i] = np.where(data['cap-surface'] == i, 1, 0)
1117        data.pop('cap-surface')
1118
1119        for i in Color._member_names_:
1120            data['cap-color-' + i] = np.where(data['cap-color'] == i, 1, 0)
1121        data.pop('cap-color')
1122
1123        data['does-bruise-or-bleed'] = [int(i == 't') for i  in data['does-
        bruise-or-bleed']]
1124
1125        for i in GillAttach._member_names_:
1126            data['gill-attachment-' + i] = np.where(data['gill-attachment'] ==
        i, 1, 0)
1127        data.pop('gill-attachment')
1128
1129        for i in GillSpacing._member_names_:
1130            data['gill-spacing-' + i] = np.where(data['gill-spacing'] == i, 1,
        0)
1131        data.pop('gill-spacing')
1132
1133        for i in Color._member_names_:
1134            data['gill-color-' + i] = np.where(data['gill-color'] == i, 1, 0)
1135        data.pop('gill-color')
1136
```

```
    for i in StemRoot._member_names_:
        data['stem-root-' + i] = np.where(data['stem-root'] == i, 1, 0)
    data.pop('stem-root')

    for i in Surface._member_names_:
        data['stem-surface-' + i] = np.where(data['stem-surface'] == i, 1,
    0)
    data.pop('stem-surface')

    for i in Color._member_names_:
        data['stem-color-' + i] = np.where(data['stem-color'] == i, 1, 0)
    data.pop('stem-color')

    for i in VeilType._member_names_:
        data['veil-type-' + i] = np.where(data['veil-type'] == i, 1, 0)
    data.pop('veil-type')

    for i in Color._member_names_:
        data['veil-color-' + i] = np.where(data['veil-color'] == i, 1, 0)
    data.pop('veil-color')

    data['has-ring'] = [int(i == 't') for i  in data['has-ring']]

    for i in RingType._member_names_:
        data['ring-type-' + i] = np.where(data['ring-type'] == i, 1, 0)
    data.pop('ring-type')

    for i in Color._member_names_:
        data['spore-print-color-' + i] = np.where(data['spore-print-color']
    == i, 1, 0)
    data.pop('spore-print-color')

    for i in Habitat._member_names_:
        data['habitat-' + i] = np.where(data['habitat'] == i, 1, 0)
    data.pop('habitat')

    for i in Season._member_names_:
        data['season-' + i] = np.where(data['season'] == i, 1, 0)
    data.pop('season')

    print("\nINITIAL VARIABLES: " + str(len(list(data.columns))))
    print("\nCOLUMNS WITH ALL 0 VALUES:\n")

    for i in (list(data.columns)):
        if(data[i] == 0).all():
            print(i)
            data.pop(i)

    print("\nTOTAL VARIABLES: " + str(len(list(data.columns))))
    print()

def createGraphMetricalValue(data, edibles, poisonous, tag, title, xLabel,
    yLabel, fileName):
```

```
1187        plt.plot(np.arange(len(data[tag][edibles])), np.sort(data[tag][edibles
        ])[::-1], label='Edible')
1188        plt.plot(np.arange(len(data[tag][poisonous])), np.sort(data[tag][
        poisonous])[::-1], c='#ff7f0e', label='Poisonous')
1189        plt.title(title)
1190        plt.xlabel(xLabel)
1191        plt.ylabel(yLabel)
1192        plt.legend()
1193        plt.savefig('../Results/Analysis/' + fileName, bbox_inches='tight')
1194        plt.close()
1195
1196    def createGraphBinaryValue(data, edibles, poisonous, tag, labels, width,
        title, fileName):
1197        x = np.arange(len(labels))  # the label locations
1198
1199        edib = [len(np.where(data[tag][edibles])[0]), len(np.where(data[tag][
        edibles] == 0)[0])]
1200        pois = [len(np.where(data[tag][poisonous])[0]), len(np.where(data[tag][
        poisonous] == 0)[0])]
1201
1202        fig, ax = plt.subplots()
1203        ax.bar(x - width/2, edib, width, label='Edible')
1204        ax.bar(x + width/2, pois, width, label='Poisonous')
1205        plt.ylabel('Mushrooms')
1206        plt.title(title)
1207        plt.xticks(x, labels)
1208        plt.legend()
1209        plt.savefig('../Results/Analysis/' + fileName, bbox_inches='tight')
1210        plt.close()
1211
1212    def createGraphEnumValue(data, edibles, poisonous, tags, labels, width,
        title, fileName):
1213        x = np.arange(len(labels))  # the label locations
1214
1215        edib = []
1216        pois = []
1217
1218        for tag in tags:
1219            edib.append(len(np.where(data[tag][edibles])[0]))
1220            pois.append(len(np.where(data[tag][poisonous])[0]))
1221
1222        fig, ax = plt.subplots(figsize=(16,9))
1223        ax.bar(x - width/2, edib, width, label='Edible')
1224        ax.bar(x + width/2, pois, width, label='Poisonous')
1225        plt.ylabel('Mushrooms')
1226        plt.title(title)
1227        plt.xticks(x, labels)
1228        plt.legend()
1229        plt.savefig('../Results/Analysis/' + fileName, bbox_inches='tight')
1230        plt.close()
1231
1232    def analyzeData(data):
1233        print(data.head())
```

```
1234
1235        print(data.describe())
1236
1237        edibles = np.where(data['class'])[0]
1238        poisonous = np.where(data['class'] == 0)[0]
1239
1240        numEdibles = len(edibles)
1241        numPoisonous = len(poisonous)
1242
1243        classMush = [numEdibles, numPoisonous]
1244
1245        # 00 number of edibles and poisonous
1246        plt.bar(np.arange(2), classMush)[1].set_color('#ff7f0e')
1247        plt.xticks(np.arange(2), ['Edibles: ' + str(len(edibles)), 'Poisonus: '
            + str(len(poisonous))])
1248        plt.ylabel('Quantity')
1249        plt.title('Number of poisonous and edibles mushrooms')
1250        plt.savefig('../Results/Analysis/00EdiblesNumb.png', bbox_inches='tight
            ')
1251        plt.close()
1252
1253        # 01 cap diameter
1254        createGraphMetricalValue(data, edibles, poisonous, 'cap-diameter',
1255          'Cap diameter of poisonous and edible mushrooms', 'Mushroom', '
            Diameter', '01CapDiameter.png')
1256
1257        # 02 cap shape
1258        createGraphEnumValue(data, edibles, poisonous,
1259        ['cap-shape-b', 'cap-shape-c', 'cap-shape-x', 'cap-shape-f', 'cap-shape
            -s', 'cap-shape-p', 'cap-shape-o'],
1260        ['Bell', 'Conical', 'Convex', 'Flat', 'Sunken', 'Spherical', 'Other'],
            0.35,
1261        'Cap shape by class', '02CapShape.png')
1262
1263        # 03 cap surface
1264        createGraphEnumValue(data, edibles, poisonous,
1265        ['cap-surface-i', 'cap-surface-g', 'cap-surface-y', 'cap-surface-s', '
            cap-surface-h', 'cap-surface-l',
1266          'cap-surface-k', 'cap-surface-t', 'cap-surface-w', 'cap-surface-e', '
            cap-surface-d'],
1267        ['Fibrous', 'Grooves', 'Scaly', 'Smooth', 'Shiny', 'Leatherly',
1268          'Silky', 'Sticky', 'Wrinkled', 'Fleshy', 'd(Unknown)'], 0.35,
1269        'Cap surface by class', '03CapSurface.png')
1270
1271        # 04 cap color
1272        createGraphEnumValue(data, edibles, poisonous,
1273        ['cap-color-n', 'cap-color-b', 'cap-color-g', 'cap-color-r', 'cap-color
            -p', 'cap-color-u',
1274        'cap-color-e', 'cap-color-w', 'cap-color-y', 'cap-color-l', 'cap-color-
            o', 'cap-color-k'],
1275        ['Brown', 'Buff', 'Gray', 'Green', 'Pink', 'Purple',
1276        'Red', 'White', 'Yellow', 'Blue', 'Orange', 'Black'], 0.35,
1277        'Cap color by class', '04CapColor.png')
```

```
1278
1279    # 05 does bruise
1280    createGraphBinaryValue(data, edibles, poisonous, 'does-bruise-or-bleed'
        ,
1281    ["Bruise or Bleed", "Doesn't"], 0.35,
1282    'Does bruise or bleed by class', '05BruiseOrBleed.png')
1283
1284    # 06 gill attachment
1285    createGraphEnumValue(data, edibles, poisonous,
1286    ['gill-attachment-empty', 'gill-attachment-a', 'gill-attachment-x', '
        gill-attachment-d',
1287    'gill-attachment-e', 'gill-attachment-s', 'gill-attachment-p', 'gill-
        attachment-f'],
1288    ['Empty value', 'Adnate', 'Adnexed', 'Decurrent',
1289    'Free', 'Sinuate', 'Pores', 'None'], 0.35,
1290    'Gill attachment by class', '06GillAttachment.png')
1291
1292    # 07 gill spacing
1293    createGraphEnumValue(data, edibles, poisonous,
1294    ['gill-spacing-empty', 'gill-spacing-c', 'gill-spacing-d', 'gill-
        spacing-f'],
1295    ['Empty value', 'Close', 'Distant', 'None'], 0.35,
1296    'Gill spacing by class', '07GillSpacing.png')
1297
1298    # 08 gill color
1299    createGraphEnumValue(data, edibles, poisonous,
1300    ['gill-color-n', 'gill-color-b', 'gill-color-g', 'gill-color-r', 'gill-
        color-p', 'gill-color-u',
1301    'gill-color-e', 'gill-color-w', 'gill-color-y', 'gill-color-o', 'gill-
        color-k', 'gill-color-f'],
1302    ['Brown', 'Buff', 'Gray', 'Green', 'Pink', 'Purple',
1303    'Red', 'White', 'Yellow', 'Orange', 'Black', 'None'], 0.35,
1304    'Gill color by class', '08GillColor.png')
1305
1306    # 09 stem height
1307    createGraphMetricalValue(data, edibles, poisonous, 'stem-height',
1308     'Stem height of poisonous and edible mushrooms', 'Mushroom', 'Height',
         '09StemHeight.png')
1309
1310    # 10 stem width
1311    createGraphMetricalValue(data, edibles, poisonous, 'stem-width',
1312     'Stem width of poisonous and edible mushrooms', 'Mushroom', 'Width', '
        10StemWidth.png')
1313
1314    # 11 stem root
1315    createGraphEnumValue(data, edibles, poisonous,
1316    ['stem-root-empty', 'stem-root-b', 'stem-root-s',
1317     'stem-root-c', 'stem-root-r', 'stem-root-f'],
1318    ['Empty value', 'Bulbous', 'Swollen',
1319     'Club', 'Rooted', 'None'], 0.35,
1320    'Stem root by class', '11StemRoot.png')
1321
1322    # 12 stem surface
```

```
1323        createGraphEnumValue(data, edibles, poisonous,
1324        ['stem-surface-i', 'stem-surface-g', 'stem-surface-y', 'stem-surface-s'
            ,
1325        'stem-surface-h', 'stem-surface-k', 'stem-surface-t', 'stem-surface-f'
            ],
1326        ['Fibrous', 'Grooves', 'Scaly', 'Smooth',
1327        'Shiny', 'Silky', 'Sticky', 'None'], 0.35,
1328        'Stem surface by class', '12StemSurface.png')
1329
1330        # 13 stem color
1331        createGraphEnumValue(data, edibles, poisonous,
1332        ['stem-color-n', 'stem-color-b', 'stem-color-g', 'stem-color-r', 'stem-
            color-p', 'stem-color-u',
1333        'stem-color-e', 'stem-color-w', 'stem-color-y', 'stem-color-l', 'stem-
            color-o', 'stem-color-k', 'stem-color-f'],
1334        ['Brown', 'Buff', 'Gray', 'Green', 'Pink', 'Purple',
1335        'Red', 'White', 'Yellow', 'Blue', 'Orange', 'Black', 'None'], 0.35,
1336        'Stem color by class', '13StemColor.png')
1337
1338        # 14 veil type
1339        createGraphEnumValue(data, edibles, poisonous,
1340        ['veil-type-empty', 'veil-type-u'],
1341        ['Partial', 'Universal'], 0.35,
1342        'Veil type by class', '14VeilType.png')
1343
1344        # 15 veil color
1345        createGraphEnumValue(data, edibles, poisonous,
1346        ['veil-color-n', 'veil-color-u', 'veil-color-e',
1347        'veil-color-w', 'veil-color-y', 'veil-color-k'],
1348        ['Brown', 'Purple','Red',
1349        'White', 'Yellow', 'Black'], 0.35,
1350        'Veil color by class', '15VeilColor.png')
1351
1352        # 16 has ring
1353        createGraphBinaryValue(data, edibles, poisonous, 'has-ring',
1354        ["Has ring", "Doesn't"], 0.35,
1355        'Does it has a ring by class', '16HasRing.png')
1356
1357        # 17 ring type
1358        createGraphEnumValue(data, edibles, poisonous,
1359        ['ring-type-empty', 'ring-type-e', 'ring-type-r', 'ring-type-g', 'ring-
            type-l',
1360        'ring-type-p', 'ring-type-z', 'ring-type-m', 'ring-type-f'],
1361        ['Empty value', 'Evanescent', 'Flaring', 'Grooved', 'Large',
1362        'Pendant', 'Zone', 'Movable', 'None'], 0.35,
1363        'Ring type by class', '17RingType.png')
1364
1365        # 18 spore print color
1366        createGraphEnumValue(data, edibles, poisonous,
1367        ['spore-print-color-n', 'spore-print-color-g', 'spore-print-color-r', '
            spore-print-color-p',
1368          'spore-print-color-u', 'spore-print-color-w', 'spore-print-color-k'],
1369        ['Brown', 'Gray', 'Green', 'Pink',
```

```
    'Purple', 'White', 'Black'], 0.35,
    'Spore print color by class', '18SporePrintColor.png')

    # 19 habitat
    createGraphEnumValue(data, edibles, poisonous,
    ['habitat-g', 'habitat-l', 'habitat-m', 'habitat-p',
    'habitat-h', 'habitat-u', 'habitat-w', 'habitat-d'],
    ['Grasses', 'Leaves', 'Meadows', 'Paths',
    'Heaths', 'Urban', 'Waste', 'Woods'], 0.35,
    'Habitat by class', '19Habitat.png')

    # 19 season
    createGraphEnumValue(data, edibles, poisonous,
    ['season-s', 'season-u', 'season-a', 'season-w'],
    ['Spring', 'Summer', 'Autumn', 'Winter'], 0.35,
    'Season by class', '20Season.png')

    # heatmap best
    plt.figure(figsize=(6, 8))
    sns.heatmap(data.corr()[['class']].sort_values('class', ascending=False
    )[1:31], annot=True, vmin=-1, vmax=1)
    plt.title('Heat map of top 30 most correlated variables to being edible
    ')
    plt.savefig('../Results/Analysis/heatMap30Best.png', bbox_inches='tight
    ')
    plt.close()

    # heatmap worst
    plt.figure(figsize=(6, 8))
    sns.heatmap(data.corr()[['class']].sort_values('class', ascending=False
    )[-30:], annot=True, vmin=-1, vmax=1)
    plt.title('Heat map of top 30 least correlated variables to being
    edible')
    plt.savefig('../Results/Analysis/heatMap30Worst.png', bbox_inches='
    tight')
    plt.close()

    # collage
    # data.hist(figsize=(10, 5))
    # plt.tight_layout()
    # plt.savefig('../Results/Analysis/collageWithAllGraphs.png',
    bbox_inches='tight')
    # plt.close()
```

src/prepareData.py

```
from threading import Thread

class ThreadWithReturnValue(Thread):
    def __init__(self, group=None, target=None, name=None,
                 args=(), kwargs={}, Verbose=None):
        Thread.__init__(self, group, target, name, args, kwargs)
        self._return = None
    def run(self):
```

```
1008            if self._target is not None:
1009                self._return = self._target(*self._args,
1010                                            **self._kwargs)
1011        def join(self, *args):
1012            Thread.join(self, *args)
1013            return self._return
```

src/threadRetVal.py

```
1000 import numpy as np
1001 import matplotlib.pyplot as plt
1002
1003 import sklearn.svm as svm
1004 from sklearn.metrics import accuracy_score
1005
1006 from pandas.io.parsers import read_csv
1007
1008 from prepareData import analyzeData, prepareData
1009 from evaluateSVM import evaluateSVM, evaluateSVMLinear
1010 from evaluateNeuronal import evalNN, getNumAcertadosNN
1011 from evaluateLogistic import evalLogisticReg, getNumAcertadosLog
1012
1013 from threadRetVal import ThreadWithReturnValue
1014
1015 import time
1016
1017 def loadCSV(fileName):
1018     data = read_csv(fileName, sep=';', on_bad_lines='skip')
1019     data.fillna('empty', inplace=True)
1020     prepareData(data)
1021     return data
1022
1023 def evalLogistic(xTrain, xVal, xTest, yTrain, yVal, yTest):
1024     print("\nCOMENCING TRAINING OF LOGISTIC REGRESION\n")
1025
1026     th, pol, acc, cost, l, exp = evalLogisticReg(xTrain, xVal, yTrain, yVal
        )
1027
1028     xPolTest = pol.fit_transform(xTest)
1029
1030     acertados = getNumAcertadosLog(th, xPolTest, yTest)
1031     accuracy = acertados*100/np.shape(xTest)[0]
1032     print('Accuracy over Test sample: ' + str(accuracy) + "%")
1033
1034     return accuracy, acc, cost, l, exp
1035
1036 def evalNueronalThread(ocultas, num_entradas, num_etiquetas, xTrain, xVal,
        xTest, yTrain, yVal, yTest, tagsTrain, tagsVal):
1037     print("\nTesting with " + str(ocultas))
1038     num_ocultas = np.array(ocultas)
1039     th, acc, cost, reg, laps = evalNN(
1040     num_entradas, num_ocultas, num_etiquetas,
1041     xTrain, xVal, yTrain, yVal, tagsTrain, tagsVal)
1042
```

```
1043        acertados = getNumAcertadosNN(xTest, yTest, th)
1044
1045        accuracy = acertados*100/np.shape(xTest)[0]
1046        print("Accuracy over Test sample: " + str(accuracy) + "%")
1047        return accuracy, acc, cost, reg, laps
1048
1049    def evalNeuronal(xTrain, xVal, xTest, yTrain, yVal, yTest, n):
1050        print("\nCOMENCING TRAINING OF NEURONAL NETWORK\n")
1051
1052        ocultas = np.array([[20], [40], [60], [80], [100], [120], [60, 20],
            [60, 40]])
1053        numOcultas = ocultas.shape[0]
1054
1055        resAcc = np.zeros(numOcultas)
1056        resAccVal = np.zeros(numOcultas)
1057        resCost = np.zeros(numOcultas)
1058        resReg = np.zeros(numOcultas)
1059        resLaps = np.zeros(numOcultas)
1060
1061        startTime = time.time()
1062
1063        tagsTrain = np.zeros((len(yTrain), 2))
1064        for i in range(len(yTrain)):
1065            tagsTrain[i][int(yTrain[i])] = 1
1066
1067        tagsVal = np.zeros((len(yVal), 2))
1068        for i in range(len(yVal)):
1069            tagsVal[i][int(yVal[i])] = 1
1070
1071        num_etiquetas = 2
1072        num_entradas = n
1073
1074        threads = np.empty(numOcultas, dtype=object)
1075
1076        for i in np.arange(numOcultas):
1077            threads[i] = ThreadWithReturnValue(target=evalNueronalThread, args
            =(ocultas[i], num_entradas, num_etiquetas, xTrain, xVal, xTest, yTrain,
             yVal, yTest, tagsTrain, tagsVal,))
1078            threads[i].start()
1079
1080        for i in np.arange(numOcultas):
1081            resAcc[i], resAccVal[i], resCost[i], resReg[i], resLaps[i] =
            threads[i].join()
1082
1083        bestAcc = np.max(resAcc)
1084        bestOcIndex = np.where(bestAcc == resAcc)[0]
1085        bestCost = resCost[bestOcIndex]
1086        bestReg = resReg[bestOcIndex]
1087        bestLaps = resLaps[bestOcIndex]
1088        bestAccVal = resAccVal[bestOcIndex]
1089
1090        print()
1091        print('Best neurons in hidden layer: ' + str(ocultas[bestOcIndex]))
```

```
1092        print('Best Accuracy: ' + str(bestAcc) + '%')
1093        print("Accuracy of train: " + str(bestAccVal) + "%")
1094        print('Best cost: ' + str(bestCost))
1095        print('Best reg: ' + str(bestReg))
1096        print('Best laps: ' + str(bestLaps))
1097
1098        endTime = time.time()
1099        print('Seconds elapsed of test: ' + str(endTime - startTime))
1100
1101        ocultasStr = []
1102        for i in np.arange(numOcultas):
1103            ocultasStr.append(str(ocultas[i]))
1104
1105        plt.plot(np.arange(numOcultas), resAcc)
1106        plt.xticks(np.arange(numOcultas), ocultasStr)
1107        plt.savefig('../Results/NeuronalNetwork/NN.png', bbox_inches='tight')
1108        plt.close()
1109
1110        return ocultas[bestOcIndex], bestAcc, bestAccVal, bestCost, bestReg,
            bestLaps
1111
1112    def evalSVM(xTrain, xVal, xTest, yTrain, yVal, yTest):
1113        print("\nCOMENCING TRAINING OF SVM\n")
1114
1115        s, acc, c, sig = evaluateSVM(xTrain, xVal, yTrain, yVal)
1116        accuracy = accuracy_score(yTest, s.predict(xTest)) * 100
1117        print('Accuracy over Test sample: ' + str(accuracy) + '%')
1118
1119        return accuracy, acc, c, sig
1120
1121    def evalSVMLinear(xTrain, xVal, xTest, yTrain, yVal, yTest):
1122        print("\nCOMENCING TRAINING OF SVM LINEAR\n")
1123
1124        s, acc, c = evaluateSVMLinear(xTrain, xVal, yTrain, yVal)
1125        accuracy = accuracy_score(yTest, s.predict(xTest)) * 100
1126        print('Accuracy over Test sample: ' + str(accuracy) + '%')
1127        return accuracy, acc, c
1128
1129    def createDataSets(x, y, m, trainPerc, valPerc, testPerc):
1130        if(trainPerc + valPerc + testPerc > 1.0):
1131            print("ERROR: Percentages given not valid")
1132            exit(-1)
1133
1134        print('Size of Training set: ' + str(int(trainPerc * m)))
1135        print('Size of Validation set: ' + str(int(valPerc * m)))
1136        print('Size of Teseting set: ' + str(int(testPerc * m)))
1137
1138        valPerc += trainPerc
1139        testPerc += valPerc
1140
1141        train = int(trainPerc * m)
1142        val = int(valPerc * m)
1143        test = int(testPerc * m)
```

```
1144
1145        xTrain = x[:train]
1146        yTrain = y[:train]
1147
1148        xVal = x[train:val]
1149        yVal = y[train:val]
1150
1151        xTest = x[val:test]
1152        yTest = y[val:test]
1153
1154        return xTrain, yTrain, xVal, yVal, xTest, yTest
1155
1156    def main():
1157        dataR = loadCSV("../Data/MushroomDataset/secondary_data_shuffled.csv")
1158
1159        #analyzeData(dataR)
1160
1161        data = dataR.to_numpy()
1162        y = data[:, 0]
1163        x = data[:, 1:]
1164
1165        m = y.shape[0]
1166        n = x.shape[1]
1167
1168        trainPerc = 0.2
1169        valPerc = 0.2
1170        testPerc = 0.2
1171
1172        # 0.2 0.2 0.2
1173        # Seconds taken for the evaluation: 8767.091146945953
1174        # Minutes taken for the evaluation: 146.11818578243256
1175        # Hours taken for the evaluation: 2.4353030963738758
1176
1177        xTrain, yTrain, xVal, yVal, xTest, yTest = createDataSets(x, y, m,
            trainPerc, valPerc, testPerc)
1178
1179        startTime = time.time()
1180
1181        logAcc, logAccVal, logCost, logL, logExp = evalLogistic(xTrain, xVal,
            xTest, yTrain, yVal, yTest)
1182
1183        nnHidenLayer, nnAcc, nnAccVal, nnCost, nnReg, nnLaps = evalNeuronal(
            xTrain, xVal, xTest, yTrain, yVal, yTest, n)
1184
1185        svmLAcc, svmLAccVal, svmLC= evalSVMLinear(xTrain, xVal, xTest, yTrain,
            yVal, yTest)
1186
1187        svmAcc, svmAccVal, svmC, svmSig = evalSVM(xTrain, xVal, xTest, yTrain,
            yVal, yTest)
1188
1189        print('\nRESULTS OF LOGISTIC REGRESSION\n')
1190        print("Best accuracy: " + str(logAcc))
1191        print("Accuracy of train: " + str(logAccVal) + "%")
```

```
1192      print('Best cost: ' + str(logCost))
1193      print('Best lambda: ' + str(logL))
1194      print('Best exponent: ' + str(logExp))
1195
1196      print('\nRESULTS OF NEURONAL NETWORK\n')
1197      print('Best Accuracy: ' + str(nnAcc) + '%')
1198      print("Accuracy of train: " + str(nnAccVal) + "%")
1199      print('Best neurons in hidden layer: ' + str(nnHidenLayer))
1200      print('Best cost: ' + str(nnCost))
1201      print('Best reg: ' + str(nnReg))
1202      print('Best laps: ' + str(nnLaps))
1203
1204      print('\nRESULTS OF SVM LINEAR\n')
1205      print("Best accuracy: " + str(svmLAcc))
1206      print("Accuracy of train: " + str(svmLAccVal) + "%")
1207      print("Best C: " + str(svmLC))
1208
1209      print('\nRESULTS OF SVM\n')
1210      print("Best accuracy: " + str(svmAcc))
1211      print("Accuracy of train: " + str(svmAccVal) + "%")
1212      print("Best C: " + str(svmC))
1213      print("Best Sigma: " + str(svmSig))
1214
1215      endTime = time.time()
1216      deltaTime = endTime - startTime
1217
1218      print()
1219      print('Seconds taken for the evaluation: ' + str(deltaTime))
1220      print('Minutes taken for the evaluation: ' + str(deltaTime/60))
1221      print('Hours taken for the evaluation: ' + str(deltaTime/3600))
1222
1223      plt.figure(figsize=(14, 10))
1224      plt.bar(np.arange(4), [logAcc, nnAcc, svmLAcc, svmAcc])
1225      plt.title('Accuracy comparasion between learning algorithms')
1226      plt.ylabel('Accuracy')
1227      xlabels = ['Logistic Regresion\nAccuracy: ' + str(logAcc) + '%', '
          Neural Network\nAccuracy :' + str(nnAcc) + '%', 'SVM Linear\nAccuracy:
          ' + str(svmLAcc) + '%', 'SVM Gaussian\nAccuracy: ' + str(svmAcc) + '%']
1228      plt.xticks(np.arange(4), xlabels)
1229      plt.savefig('../Results/FinalGraph.png', bbox_inches='tight')
1230      plt.close()
1231
1232  if __name__ == "__main__":
1233      main()
```

src/main.py