MOBILE PRICE CLASSIFICATION

Proyecto final de Aprendizaje Automático y Minería de Datos

Presentado por

Sergio Gavilán Fernández sgavil01@ucm.es

Alejandro Villar Rubio alvill04@ucm.es

Facultad de Informática, Universidad Complutense de Madrid

Grado en Desarrollo de Videojuegos

Madrid, 2019/2020

Contenido

Resumen

La elección de un teléfono móvil puede ser en algunas ocasiones un quebradero de cabeza para algunas personas, ya sea por su edad o simplemente porque no conocen la industria lo suficiente. Además, a este desconocimiento hay que añadir que cada día hay más dispositivos en el mercado por lo que la dificultad de su compra aumenta. Esto puede provocar la llamada **paradoja de la elección** ("La paradoja de la elección: por qué nos cuesta decidir," n.d.), nuestra tendencia a estar menos satisfechos con nuestras adquisiciones mientras más alternativas existan.

En este proyecto se planteará una herramienta con la cuál cualquier persona podrá crear un teléfono móvil a su gusto y comprobar en qué rango de precio se encuentra. Con esto, el usuario podrá orientarse para su posterior compra.

Objetivos

El objetivo principal de este proyecto es, como se ha nombrado anteriormente, proporcionar una herramienta con la que poder comprobar el rango de precio de un determinado teléfono móvil.

Para ello se aplicarán diversas técnicas de clasificación sobre la base de datos que se dispone y realizar una selección para usar la que proporcione una mayor precisión.

Las técnicas que se usarán son:

- Regresión logística.
- Redes neuronales.
- Support Vector Machines (SVM).

Además, *regresión logística* y *redes neuronales* se realizarán con una librería de *Deep learning* para Python, **Keras** ("Home - Keras Documentation," n.d.).

Trabajo previo

Se dispone de una base de datos adquirida de la plataforma *Kaggle* ("Mobile Price Classification | Kaggle," n.d.). Esta proporciona dos archivos, **test.csv** y **train.csv**, con un total de 21 características que conforman un teléfono móvil.

Capacidad batería	Bluetooth	Velocidad del procesador
Dual Sim	Resolución cámara frontal	4G
Memoria interna	Profundidad del teléfono móvil (cm)	Peso
Número de núcleos del procesador	Resolución de la cámara principal	Resolución de la pantalla, altura

Resolución de la pantalla, anchura	RAM	Altura
Anchura	Autonomía de la batería	3G
Pantalla táctil	Wifi	Rango de precio / ID

Hay una pequeña diferencia entre estos dos archivos, ambos tienen el mismo número de características, pero *train* tiene una columna con "price_range" y *test* una con "id". Posiblemente este último no tiene la información del precio porque en alguna ocasión se ha usado este conjunto de datos para alguna competición de **machine learning**. Esto hace que solo se pueda usar el archivo *train* porque es el que contiene lo que será la futura "Y".

El conjunto de datos *train* está formado por 21 características y 2000 ejemplos. Esto hace que se pueda dividir en 3 tipos de conjuntos y aun así tener un gran número de ejemplos en cada uno. Las diferentes divisiones son: **train** (60%, 1200 ejemplos)**, validation** (20%, 400 ejemplos) y **test** (20%, 400 ejemplos). Esta división se ha realizado a través de código (ver **Apéndice A**).

De las 21 características mencionadas anteriormente, se cogerán 20 de ellas para estudiar la restante, el **rango de precio**. Esta característica puede adquirir un valor del 0 al 3, ambos incluidos, donde 0 corresponderá a la gama de teléfonos más baja y 3 a la gama más alta. Al tener estas características unos rangos numéricos tan diversos se han normalizado estos datos en todos los experimentos.

Nota: *Se proporciona una vista de la estructura de directorios y ficheros por si se quiere probar el proyecto (ver* **Apéndice C***)*

Herramientas usadas

Para realizar este trabajo se han utilizado diversas librerías que se han estudiado durante el curso:

- **Numpy**. Es un paquete que permite manejar de forma eficiente contenedores multidimensionales de datos genéricos. ("NumPy — NumPy," n.d.)
- **Matplotlib**. Permite trazar gráficas 2D. ("Matplotlib: Python plotting — Matplotlib 3.1.2 documentation," n.d.)
- **SciPy**. Ecosistema basado en Python de código abierto para matemáticos, científicos e ingenieros. ("SciPy.org — SciPy.org," n.d.)
- **Scikit-learn**. Librería de código abierto de aprendizaje automático que aporta herramientas para el aprendizaje supervisado y no supervisado.

Regresión Logística

Implementación

En primer lugar, se han preparado los siguientes datos y funciones para que los cálculos sean correctos:

- Con la función *num_to_vector* se ha creado un nuevo vector Y donde cada elemento está marcado como "0" o "1" dependiendo si coincide con el valor requerido, es decir, si se pide que se haga con el valor "2" pondrá todos los elementos a "0" excepto los que valen "2" en la Y, que los pondrá a "1".

Además, se ha creado una función para averiguar el mejor valor para *lambda*, de los valores proporcionados en un vector ([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3]), del algoritmo que se está aplicando en ese momento. Estos cálculos necesitan los dataset *train* y *validation.* El resultado final es la *lambda* que menor coste ha resultado tener con el dataset *validation*. Esto es debido a que si se elige el menor proporcionado por *train* habría una gran probabilidad de que se produjera sobreajuste.

A parte de estas funcionalidades que se han nombrado se encuentra la función *get_optimize_theta* al igual que en regresión, la cual devuelve los pesos óptimos según los valores proporcionados.

Se ha implementado también lo que se podría decir que es la función principal, *oneVsAll*. Esta entrena varios clasificadores por *regresión logística* con un determinado término de regularización y devuelve el resultado en una matriz, donde la fila i-ésima corresponde al clasificador de la etiqueta i-ésima. Con etiqueta se refiere a los valores que puede tomar Y ([0, 3]).

Queda añadir que se han usado un total de 5 algoritmos: BGFS, CG, L-BFGS-B, SLSQP y TNC. Se han excluido los demás algoritmos debido al tiempo que ocupa hacer tantos cada vez que se quiere mostrar una gráfica.

Resultados

Se pueden observar tres tipos de gráficas distintas:

- El porcentaje de error que tienen los valores de las distintas *lambdas* de cada algoritmo (ver **Apéndice B**). Lo más llamativo es que a medida que aumenta el valor de *lambda* más error de clasificación se obtiene.
- La precisión que tienen los algoritmos con cada una de las *lambdas* (ver **Apéndice B**). Se puede observar que no hay mucha diferencia de precisión entre los distintos valores, pero destaca que, a mayor *lambda*, menor precisión.

- Precisión de los algoritmos con el mejor valor de *lambda* respectivamente. Las precisiones son muy similares, se podría descartar estas diferencias y escoger cualquiera.

Nota: *Es posible que los resultados no coincidan entre las diversas gráficas debido a que son distintas ejecuciones.*

Redes Neuronales

Implementación

En primer lugar, se han preparado los siguientes datos y funciones para que los cálculos sean correctos:

- Con la función *num_to_vector* se ha creado una nueva matriz Y donde cada vector está formado por "0" exceptuando el valor marcado en la Y del conjunto de datos, que se pone a "1".
- La función *random_weight(L_in, L_out)* crea dos matrices de pesos de manera aleatoria de tamaño (L_out, L_in +1).
- El tamaño de las capas, donde la primera es 20, número de características; la segunda se ha decidido que sea 8 y la tercera es 4, valor máximo + 1 que puede alcanzar la Y.

En segundo lugar, se ha implementado las funciones coste y gradiente de la red neuronal. Luego se ha creado la función *backprop* que utiliza la nueva matriz Y, las matrices de pesos dados aleatoriamente y la propagación hacia delante para calcular el coste regularizado y el gradiente.

Además, como en *regresión logística* se ha creado una función para averiguar el mejor valor para *lambda* del algoritmo que se está aplicando en ese momento. Estos cálculos necesitan los dataset *train* y *validation.*

A parte de estas funcionalidades que se han nombrado se encuentra la función *get_optimize_theta* al igual que en regresión, la cual devuelve los pesos óptimos según los valores proporcionados.

Una vez que se han implementado todo lo que se ha comentado (ver **Apéndice A**), los pasos a seguir son los siguientes:

1. Se usa la función *num_vector* para crear las matrices de "Y" e "Y_val".
2. Se asignan valores aleatorios a dos matrices de pesos, para ello está *random_weight.*
3. Se concatenan los pesos en un solo vector.
4. Se averigua el valor de *lambda* que proporciona la mejor precisión para respectivo algoritmo.
5. Se averiguan los pesos óptimos según la información que hay hasta el momento.

6. Por último, se averigua el coste correspondiente usando "X_test" y los pesos óptimos. Además, se calcula la precisión de los cálculos mediante "Y_test".

Se han usado el mismo conjunto de algoritmos que en *regresión logística* para dar uniformidad a los resultados.

Resultados

Se han extraído tres tipos de gráficas:

- El porcentaje de error que tienen los valores de las distintas *lambdas* de cada algoritmo (ver **Apéndice B**). Lo más llamativo es que a medida que aumenta el valor de *lambda* más error de clasificación se obtiene.
- La precisión que tienen los algoritmos con cada una de las *lambdas* (ver **Apéndice B**). Se puede observar que no hay mucha diferencia de precisión entre los distintos valores, las mayores fluctuaciones las sufre el algoritmo TNC.
- Precisión de los algoritmos con el mejor valor de *lambda* respectivamente, modificando el tamaño de la capa oculta (ver **Apéndice B**). Las precisiones son muy similares, se podría descartar estas diferencias y escoger cualquiera, pero se puede observar que hay dos algoritmos que han resultado ser los mejores, BFGS y L-BFGS-B, ambos con un tamaño de capa 8.

Nota: *Las gráficas, a excepción del último punto, han sido dibujadas con una capa oculta de tamaño 8 como se ha dicho previamente.*

Nota: *Es posible que los resultados no coincidan entre las diversas gráficas debido a que son distintas ejecuciones.*

Support Vector Machines

Implementación

Se han utilizado los siguientes *kernels* para las SVM: Sigmoide, Gaussiano, Polinómico y Lineal. El proceso ha consistido en crear una máquina con cada *kernel* correspondiente para ajustar el entrenamiento a los parámetros que necesita cada una:

- **Sigmoide**: Se ha buscado el valor *C* más adecuado sin sobre ajustar ni tener sesgo en el modelo y hemos observado que el valor del grado no modificaba nuestros resultados.
- **Gaussiano**: De la misma manera se ha buscado el *C* más adecuado.

- **Polinómico**: En este caso se ha comprobado los resultados con diversos *grados* del polinomio.
- **Lineal**: Por último, en este modelo se ha aplicado el mismo procedimiento que en el sigmoide y el gaussiano para encontrar el valor *C* que mejor se ajusta al modelo.

En todos estos casos se ha comprobado el resultado de los distintos ajustes de las máquinas con el error que se obtiene prediciendo con dichas *SVM* a los ejemplos de validación y los de entrenamiento. Después de implementar todo esto (ver **Apéndice A**) se han ajustado las máquinas a los valores óptimos de cada *kernel* y se han probado frente a los ejemplos de entrenamiento para comprobar la efectividad de cada *SVM* en el modelo de datos.

Resultados

Hay un total de cuatro tipos de gráficas:

- Matriz de confusión donde se puede ver el desempeño de cada *kernel* que se emplea ha empleado (ver **Apéndice B**)
- Búsqueda de los valores de *C* que mejor se adecuan en cada caso (ver **Apéndice B**).
- Frontera de decisión para ver cómo se ajustan los valores del *kernel* a los ejemplos de entrenamiento, ya que se partía de 20 elementos se ha utilizado la función *f_classif* de *scikit-learn* para reducir los atributos a los 2 más significativos obteniendo una visión aproximada de cómo se ajustan las fronteras de decisión para cada clasificador (ver **Apéndice B**).
- Desempeño general de las distintas *SVM* (ver **Apéndice B**).

Nota: *Es posible que los resultados no coincidan entre las diversas gráficas debido a que son distintas ejecuciones.*

Keras

En este apartado se explicarán tanto *regresión logística* y *redes neuronales*, esto se debe a que *Keras* es una API de alto nivel para redes neuronales, pero el funcionamiento de la red más básica es en realidad el mismo que el de regresión logística, por lo que realizando algunos cambios se pueden implementar las dos funcionalidades (ver **Apéndice A**).

La única diferencia que existe entre ambas implementaciones es el número de capas, ya que en redes se necesitan tres capas (para mantener la cohesión entre los apartados anteriores) y en regresión solo se necesita una. Se puede ver como en la creación de estas para redes, solo la última tiene el factor de **activation** como *sigmoid*, ya que no es buena idea poner esta en las capas ocultas.

En el apartado de la compilación se puede ver que el parámetro **loss** es distinto al que se ha mostrado en clase. Esto se debe a que *binary_crossentropy* se usa, como su nombre indica, para una solución

binaria, es decir, 0 o 1, pero en este proyecto se necesitan un total de cuatro posibles soluciones. Así que esto deja la posibilidad a dos valores distintos para *loss: categorical_crossentropy* y *sparse_categorical_crossentropy*. Ambos sirven para cuando se está trabajando sobre clasificaciones multiclase. Se ha optado por usar el segundo de estos dos debido a que las soluciones son enteros, a diferencia del primero en el que las soluciones deberían ser vectores donde se usa *One Hot Encoding*.

Una vez implementado, se han sacado las respectivas gráficas. Aquí se ha decidido mostrar los distintos tipos de gráficas: las de **precisión** y las de **pérdidas**. En estas se pueden ver diversos comportamientos ya que se ha decidido usar el vector de lambdas que ya se ha utilizado en las implementaciones anteriores y unas gráficas en las que *Keras* decide por defecto que valor le da (ver **Apéndice B**).

Phone CompluHouse

Funcionalidad

Como se nombró al principio del documento, se ha intentado crear una aplicación web para darle un uso a estos conocimientos que han sido aplicados en los apartados anteriores.

Tiene una funcionalidad muy básica, simplemente el usuario tiene que elegir las características del móvil que quiere comprar y la aplicación le dirá en qué gama de precios se encuentra: *GAMA BAJA, BAMA MEDIA, GAMA ALTA* y *GAMA SUPERIOR*.

Implementación

Los datos a tener en cuenta para comprender su implementación es que se ha usado *redes neuronales* debido a que como se puede observar en sus gráficas, es la técnica de aprendizaje automático que mejores resultados ha ofrecido. Además, se ha elegido el algoritmo *TNC* debido a la velocidad de ejecución, su uso constante durante la asignatura y la poca diferencia de precisión con los demás.

Teniendo en cuenta estas decisiones solo queda nombrar los pasos seguidos (ver **Apéndice A**):

1. Se guardan los pesos óptimos que se calculan con el algoritmo TNC en redes neuronales.
2. Se cargan los datos de un dataset ya creado anteriormente, en este caso se ha decidido usar *test* de forma aleatoria. Luego se extrae la supuesta X.
3. Una vez que el usuario introduce las características de su teléfono deseado, se guarda en un array y luego se añade como nueva fila al dataset del punto anterior. Esto se hace para poder tener una gran variedad de ejemplos con los que poder normalizar la matriz X.
4. Una vez que se tienen los datos preparados se hace una pasada hacia delante para calcular la solución que espera el usuario.
5. Por último, se almacena en un archivo de texto la gama a la que pertenece dependiendo del valor calculado.

Nota: *No se ha explicado la implementación de la aplicación web debido a que no está en las competencias de la asignatura. De todas formas, se proporcionará el código por si se quiere probar.*

Lanzar la aplicación

Para probar el funcionamiento de esta aplicación es necesario extraer los pesos óptimos; organizar la estructura de ficheros y directorios correctamente, y lanzar la aplicación desde la consola de comandos.

1. Para extraer los pesos óptimos hay que descomentar la línea de *savemat* que se puede ver en el código de *redes neuronales*. Se pueden los pesos de cualquier algoritmo, pero para la implementación se ha usado los del *TNC*.
2. La estructura de directorios y ficheros es algo especial debido a que se están usando servidores para poder lanzar la aplicación (ver **Apéndice C**).
3. Para lanzar la aplicación es necesario ejecutar el archivo *run.py*. Después de esto, se podrá ver en consola una URL, la cual hay que usar en el navegador para poder acceder a la aplicación.

Conclusiones

Finalmente se han conseguido realizar todos los objetivos propuestos en este proyecto y expresar de la mejor forma posible todos los conocimientos adquiridos durante la asignatura, añadiendo una aplicación en la que se puede mostrar una sencilla, pero posible funcionalidad que tiene el aprendizaje automático en el día a día de las personas.

Bibliografía

Home - Keras Documentation. (n.d.). Retrieved January 21, 2020, from https://keras.io/

La paradoja de la elección: por qué nos cuesta decidir. (n.d.). Retrieved December 24, 2019, from https://hipertextual.com/2015/07/paradoja-eleccion

Matplotlib: Python plotting — Matplotlib 3.1.2 documentation. (n.d.). Retrieved January 10, 2020, from https://matplotlib.org/

Mobile Price Classification | Kaggle. (n.d.). Retrieved December 24, 2019, from https://www.kaggle.com/iabhishekofficial/mobile-price-classification

NumPy — NumPy. (n.d.). Retrieved January 10, 2020, from https://numpy.org/

SciPy.org — SciPy.org. (n.d.). Retrieved January 10, 2020, from https://www.scipy.org/

Apéndices

Apéndice A. Código utilizado

Código usado para dividir el dataset principal

datasetPreparation.py

```
from pandas.io.parsers import read_csv
import csv
import numpy as np


def carga_csv(file_name):
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores[1:]


def main():
    datos = carga_csv("train.csv")
    # Eliminamos la primera fila con los atributos del dataset

    datosLen = len(datos)  # 2001 casos de entrenamiento

    # Cogemos el 60% de los datos set de entrenamiento
    n_training = int(datosLen*0.6)

    training_set = datos[:n_training]

    # Por otro lado el 20% para el set de validacion
    n_validation = int(datosLen*0.2)

    validation_set = datos[n_training:n_training+n_validation]

    # Por ultimo el 20% restante para el conjunto de prueba
    n_test = datosLen - n_training - n_validation

    test_set = datos[-n_test:]

    with open('ProcessedDataSet/train.csv', mode='w', newline='') as processedTrainin
g:
        processedTrainingWriter = csv.writer(
            processedTraining, delimiter=',')
        processedTrainingWriter.writerows(training_set)

    with open('ProcessedDataSet/validation.csv', mode='w', newline='') as processedVa
lidation:
        processedValidationWriter = csv.writer(
            processedValidation, delimiter=',')
        processedValidationWriter.writerows(validation_set)

    with open('ProcessedDataSet/test.csv', mode='w', newline='') as processedTest:
        processedTestWriter = csv.writer(processedTest, delimiter=',')
        processedTestWriter.writerows(test_set)


main()
```

Código usado para Regresión Logística

reg_logistica.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
import math


#########################################################################
###############             FUNCIONES BASICAS          #################
#########################################################################

""" Sigmoide """

def sigmoid(z):
    sigmoid = 1 / (1 + np.exp(-z))
    return sigmoid


""" Calcula el coste de un determinado conjunto de ejemplos """

def f_cost(Theta, X, Y, reg):
    m = X.shape[0]
    h_theta = sigmoid(np.dot(X, Theta))

    # Calculo del coste sin el termino de regularizacion
    term1 = np.dot(-Y.T, np.log(h_theta))
    term2 = np.dot((1 - Y).T, np.log(1 - h_theta))

    # Calculo del termino de regularizacion
    reg_term = (reg / (2 * m)) * np.sum(np.square(Theta[1:]))

    # Calculo del coste
    cost = (np.sum(term1 - term2) / m) + reg_term

    return cost


""" Calcula el gradiente de un determinado conjunto de ejemplos """

def f_gradient(Theta, X, Y, reg):
    m = X.shape[0]
    h_theta = sigmoid(np.dot(X, Theta))

    # Calculo del gradiente sin el termino de regularizacion
    reg_term = (reg / m) * (Theta[1:])

    # Calculo del termino de regularizacion
    gradient = (1 / m) * np.dot(X.T, (h_theta - Y))

    # Calculo del gradiente
    gradient[1:] = gradient[1:] + reg_term
```

```
    return gradient


""" Devuelve el coste y el gradiente """

def f_opt(Theta, X, Y, reg):
    return f_cost(Theta, X, Y, reg), f_gradient(Theta, X, Y, reg)


#########################################################################
#############   FUNCIONES USADAS PARA EL ENTRENAMIENTO   ################
#########################################################################

def num_to_vector(Y, n):
    newY = np.array((Y == n) * 1)
    newY = newY[:None]

    return newY

""" Calcula el Theta optimo """

def get_optimize_theta(X, Y, reg, comp_method, use_jac):
    initial_theta = np.zeros((X.shape[1], 1))

    if use_jac:
        optTheta = minimize(fun=f_cost, x0=initial_theta,
                            args=(X, Y, reg), method=comp_method, jac=f_gradient)
    else:
        optTheta = minimize(fun=f_cost, x0=initial_theta,
                            args=(X, Y, reg), method=comp_method)

    return optTheta.x


""" Selecciona el mejor termino de regularizacion de una tupla de posibles valores ""
"

def lambda_term_selection(X, Y, X_val, Y_val, comp_method, use_jac):
    lambda_vec = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3])

    error_train = np.zeros((len(lambda_vec), 1))
    error_val = np.zeros((len(lambda_vec), 1))

    for i in range(len(lambda_vec)):
        reg = lambda_vec[i]

        for n in range(4):
            newY = num_to_vector(Y, n)
            newY_val = num_to_vector(Y_val, n)

            Theta = get_optimize_theta(X, newY, reg, comp_method, use_jac)

            error_train[i] += f_opt(Theta, X, newY, reg)[0]
            error_val[i] += f_opt(Theta, X_val, newY_val, reg)[0]

    draw_lambda_values(lambda_vec, error_train, error_val, method=comp_method)
    best_lambda = 0
```

```
    min_error = float("inf")

    for i in range(len(lambda_vec)):
        if not math.isnan(error_val[i]) and error_val[i] < min_error:
            min_error = error_val[i]
            best_lambda = lambda_vec[i]

    return best_lambda


""" Entrena los clasificadores de cada clase """

def oneVsAll(X, Y, num_of_price_range, reg, comp_method, use_jac):
    # Numero de propiedades de los ejemplos
    n = X.shape[1]

    matResult = np.zeros((num_of_price_range, n))  # (4, 21)

    for i in range(num_of_price_range):
        # Se obtiene una nueva "y" donde se indica si el ejemplo
        # j-esimo pertence a dicha clase o no.
        newY = num_to_vector(Y, i)

        matResult[i] = get_optimize_theta(
            X, newY, reg, comp_method, use_jac).ravel()

    return matResult


""" Calcula la precision """

def testClassificator(Theta, X, Y):
    aciertos = 0
    for m in range(X.shape[0]):  # Para cada ejemplo de entrenamiento
        bestClassificator = -1
        index = 0
        for j in range(Theta.shape[0]):  # Ponemos a prueba cada clasificador
            result = sigmoid(np.dot(Theta[j], X[m]))
            if(result > bestClassificator):
                bestClassificator = result
                index = j

        if(index == Y[m]):
            aciertos += 1

    precission = round((aciertos / X.shape[0]) * 100, 1)

    return precission


def draw_lambda_values(lambda_values, error_train, error_val, method):
    plt.figure(figsize=(8, 5))
    plt.plot(lambda_values, error_val, 'or--', label='Validation Set Error')
    plt.plot(lambda_values, error_train, 'bo--', label='Training Set Error')
    plt.xlabel('$\lambda$ value', fontsize=16)
    plt.ylabel('Classification Error [%]', fontsize=14)
    plt.title(f'Finding Best $\lambda$ value for method {method}', fontsize=18)
    plt.xscale('log')
```

```
    plt.legend()
    plt.show()


def logistic_regression(X, Y, X_val, Y_val, X_test, Y_test, method, jac):
    best_lambda = lambda_term_selection(X, Y, X_val, Y_val, method, jac)
    optTheta = oneVsAll(X, Y, 4, best_lambda, method, jac)
    print(f'método {method} terminado con éxito!')
    return testClassificator(optTheta, X_test, Y_test)
```

main.py

```
from matplotlib.ticker import FuncFormatter
import matplotlib.pyplot as plt

import numpy as np
from pandas.io.parsers import read_csv
import reg_logistica

'''
    Theta: (n + 1, 1)
    X: (m, n + 1)
    Y: (m, 1)
'''


def draw_precission(precission):
    plt.figure(figsize=(14, 6))
    plt.title('Regularized Logistic Regression with $\lambda$ = 1')
    plt.xlabel('Algorithm method')
    plt.ylabel('Precission')

    plt.ylim(0, 100)

    x = np.arange(len(precission))
    rects = plt.bar(x, precission, color='red')
    plt.xticks(x, ('CG', 'BFGS', 'L-BFGS-B', 'TNC', 'SLSQP'))
    for rect in rects:
        height = rect.get_height()
        plt.annotate('{}%'.format(height),
                     xy=(rect.get_x() + rect.get_width() / 2, height / 2),
                     xytext=(0, 3),  # 3 points vertical offset
                     textcoords="offset points",
                     ha='center', va='bottom', color=(1.0, 1.0, 1.0, 1.0),
                    fontsize=20, weight='bold')
    plt.show()

# Carga un fichero ".csv" y devuelve los datos


def load_data(file_name):
    values = read_csv(file_name, header=None).values
    return values.astype(float)


def normalize_matrix(X):
```

```
    mu = np.mean(X, axis=0)
    X_norm = X - mu

    sigma = np.std(X_norm, axis=0)
    X_norm = X_norm / sigma

    return X_norm


def get_data_matrix(data):
    X = np.delete(data, data.shape[1] - 1, axis=1)  # (1200, 20)
    X = normalize_matrix(X)
    X = np.insert(X, 0, 1, axis=1)  # (1200, 21)
    Y = data[:, data.shape[1] - 1]  # (1200,)

    return X, Y


def main():
    train_data = load_data("../ProcessedDataSet/train.csv")
    validation_data = load_data("../ProcessedDataSet/validation.csv")
    test_data = load_data("../ProcessedDataSet/test.csv")

    X, Y = get_data_matrix(train_data)
    X_val, Y_val = get_data_matrix(validation_data)
    X_test, Y_test = get_data_matrix(test_data)

    cg_precission = reg_logistica.logistic_regression(
        X, Y, X_val, Y_val, X_test, Y_test, 'CG', True)
    bfgs_precission = reg_logistica.logistic_regression(
        X, Y, X_val, Y_val, X_test, Y_test, 'BFGS', True)
    l_bfgs_b_precission = reg_logistica.logistic_regression(
        X, Y, X_val, Y_val, X_test, Y_test, 'L-BFGS-B', True)
    tnc_precission = reg_logistica.logistic_regression(
        X, Y, X_val, Y_val, X_test, Y_test, 'TNC', True)
    slsqp_precission = reg_logistica.logistic_regression(
        X, Y, X_val, Y_val, X_test, Y_test, 'SLSQP', True)

    precission = [cg_precission, bfgs_precission, l_bfgs_b_precission, tnc_precission
, slsqp_precission]

    draw_precission(precission)

main()
```

Código usado para Redes Neuronales

neural_network.py

```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
import math
from scipy.io import savemat

# Función sigmoide
```

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Cálculo de la derivada de la función sigmoide
def der_sigmoid(z):
    return (sigmoid(z) * (1.0 - sigmoid(z)))

# Cáculo del coste no regularizado
def coste_no_reg(m, h, y):
    J = 0
    for i in range(m):
        J += np.sum(-y[i] * np.log(h[i]) \
             - (1 - y[i]) * np.log(1 - h[i]))
    return (J / m)


# Cálculo del coste regularizado
def f_cost(m, h, Y, reg, theta1, theta2):
    return (coste_no_reg(m, h, Y) +
        ((reg / (2 * m)) *
        (np.sum(np.square(theta1[:, 1:])) +
        np.sum(np.square(theta2[:, 1:])))))


# Inicializa una matriz de pesos aleatorios
def random_weight(L_in, L_out):
    ini = 0.12
    theta = np.random.uniform(low=-ini, high=ini, size=(L_out, L_in))

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

    return theta

def num_to_vector(n, output_layer):
    lenN = len(n)
    n = n.ravel()
    n_onehot = np.zeros((lenN, output_layer))
    for i in range(lenN):
        n_onehot[i][int(n[i])] = 1

    return n_onehot


""" Selecciona el mejor termino de regularizacion de una tupla de posibles valores ""
"
def lambda_term_selection(nn_params, input_layer, hidden_layer , output_layer, X, Y_o
nehot, \
    X_val, Y_val_onehot, comp_method, use_jac):
    lambda_vec = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])

    error_train = np.zeros((len(lambda_vec), 1))
    error_val = np.zeros((len(lambda_vec), 1))

    for i in range(len(lambda_vec)):
        reg = lambda_vec[i]

        Theta = get_optimize_theta(nn_params, input_layer, hidden_layer, \
        output_layer, X, Y_onehot, reg, comp_method, use_jac)
```

```
        # Despliegue de params_rn para sacar las Thetas
        theta1 = np.reshape(Theta.x[:hidden_layer * (input_layer + 1)],
                (hidden_layer, (input_layer + 1)))

        theta2 = np.reshape(Theta.x[hidden_layer * (input_layer + 1): ],
            (output_layer, (hidden_layer + 1)))

        a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)
        error_train[i] = f_cost(X.shape[0], h, Y_onehot, reg, theta1, theta2)

        a1, z2, a2, z3, h_val = forward_propagate(X_val, theta1, theta2)
        error_val[i] = f_cost(X_val.shape[0], h_val, Y_val_onehot, reg, theta1, theta
2)

    #draw_lambda_values(lambda_vec, error_train, error_val, method=comp_method)
    best_lambda = 0
    min_error = float("inf")

    for i in range(len(lambda_vec)):
        if not math.isnan(error_val[i]) and error_val[i] < min_error:
            min_error = error_val[i]
            best_lambda = lambda_vec[i]

    return best_lambda


def get_optimize_theta(nn_params, input_layer, hidden_layer , output_layer, X, Y_oneh
ot, reg, comp_method, use_jac):
    initial_theta = np.zeros((X.shape[1], 1))

    # Obtención de los pesos óptimos entrenando una red con los pesos aleatorios
    if use_jac:
        optTheta = opt.minimize(
            fun=backprop,
            x0=nn_params,
            args=(input_layer, hidden_layer, output_layer, X, Y_onehot, reg),
            method=comp_method,
            jac=True,
            options={'maxiter': 70})
    else:
        optTheta = opt.minimize(
            fun=backprop,
            x0=nn_params,
            args=(input_layer, hidden_layer, output_layer, X, Y_onehot, reg),
            method=comp_method,
            options={'maxiter': 70})

    return optTheta

# Devuelve "Y" a partir de una X y no unos pesos determinados
def forward_propagate(X, theta1, theta2):
    m = X.shape[0]

    a1 = np.hstack([np.ones([m, 1]), X])    # (5000, 401)
    z2 = np.dot(a1, theta1.T)   # (5000, 25)

    a2 = np.hstack([np.ones([m, 1]), sigmoid(z2)])  # (5000, 26)
```

```
    z3 = np.dot(a2, theta2.T)   # (5000, 10)

    h = sigmoid(z3) # (5000, 10)

    return a1, z2, a2, z3, h

# Devuelve el coste y el gradiente de una red neuronal de dos capas
def backprop(params_rn, input_layer, hidden_layer, output_layer, X, y, reg):
    m = X.shape[0]

    # Despliegue de params_rn para sacar las Thetas
    theta1 = np.reshape(params_rn[:hidden_layer * (input_layer + 1)],
            (hidden_layer, (input_layer + 1)))

    theta2 = np.reshape(params_rn[hidden_layer * (input_layer + 1): ],
        (output_layer, (hidden_layer + 1)))

    a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)

    coste = f_cost(m, h, y, reg, theta1, theta2) # Coste regularizado

    # Inicialización de dos matrices "delta" a 0 con el tamaño de los thethas respect
ivos
    delta1 = np.zeros_like(theta1)
    delta2 = np.zeros_like(theta2)

    # Por cada ejemplo
    for t in range(m):
        a1t = a1[t, :] # (1, 401)
        a2t = a2[t, :] # (1, 26)
        ht = h[t, :] # (1, 10)
        yt = y[t]

        d3t = ht - yt
        d2t = np.dot(theta2.T, d3t) * (a2t * (1 - a2t)) # (1, 26)

        delta1 = delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
        delta2 = delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])

    delta1 = delta1 / m
    delta2 = delta2 / m

    # Gradiente perteneciente a cada delta
    delta1[:, 1:] = delta1[:, 1:] + (reg * theta1[:, 1:]) / m
    delta2[:, 1:] = delta2[:, 1:] + (reg * theta2[:, 1:]) / m

    # Concatenación de los gradientes
    grad = np.concatenate((np.ravel(delta1), np.ravel(delta2)))

    return coste, grad

# Cálculo de la precisión
def testClassificator(h, Y):
    aciertos = 0
    for i in range (h.shape[0]):
        max = np.argmax(h[i])

        if max == Y[i]:
```

```
            aciertos += 1

    precision = round((aciertos / h.shape[0]) * 100, 1)
    return precision

def training_neural_network(X, Y, X_val, Y_val, X_test, Y_test, input_layer, hidden_l
ayer, output_layer, \
    comp_method, use_jac):
    # Transforma Y en un vector
    Y_onehot = num_to_vector(Y, output_layer)
    Y_val_onehot = num_to_vector(Y_val, output_layer)

    # Inicialización de dos matrices de pesos de manera aleatoria
    Theta1 = random_weight(input_layer, hidden_layer)
    Theta2 = random_weight(hidden_layer, output_layer)

    # Crea una lista de Thetas
    Thetas = [Theta1, Theta2]

    # Concatenación de las matrices de pesos en un solo vector
    unrolled_Thetas = [Thetas[i].ravel() for i,_ in enumerate(Thetas)]
    nn_params = np.concatenate(unrolled_Thetas)

    reg = lambda_term_selection(nn_params, input_layer, hidden_layer, output_layer, \
        X, Y_onehot, X_val, Y_val_onehot, comp_method, use_jac)

    optTheta = get_optimize_theta(nn_params, input_layer, hidden_layer, \
        output_layer, X, Y_onehot, reg, comp_method, use_jac)

    # Desglose de los pesos óptimos en dos matrices
    newTheta1 = np.reshape(optTheta.x[:hidden_layer * (input_layer + 1)],
        (hidden_layer, (input_layer + 1)))

    newTheta2 = np.reshape(optTheta.x[hidden_layer * (input_layer + 1): ],
        (output_layer, (hidden_layer + 1)))

    #savemat("weights.mat", {'Theta1': newTheta1, 'Theta2': newTheta2})

    # H, resultado de la red al usar los pesos óptimos
    a1, z2, a2, z3, h = forward_propagate(X_test, newTheta1, newTheta2)

    # Cálculo de la precisión
    return testClassificator(h, Y_test)

def draw_lambda_values(lambda_values, error_train, error_val, method):
    plt.figure(figsize=(8, 5))
    plt.plot(lambda_values, error_val, 'or--', label='Validation Set Error')
    plt.plot(lambda_values, error_train, 'bo--', label='Training Set Error')
    plt.xlabel('$\lambda$ value', fontsize=16)
    plt.ylabel('Classification Error [%]', fontsize=14)
    plt.title(f'Finding Best $\lambda$ value for method {method}', fontsize=18)
    plt.xscale('log')
    plt.legend()
    plt.show()
```

main.py

```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt
import neural_network

'''

3 capas:
    + 20 en la primera capa (la primera siempre fijada +1)
    + 8 en la capa oculta
    + 4 en la de salida

Theta1 de dimension (8 x 21)
Theta2 de dimension (4 x 9)

'''

def draw_precission(precission):
    plt.figure(figsize=(14, 6))
    plt.title('Neural Network Precission with the best $\lambda$ for each method')
    plt.xlabel('Algorithm method')
    plt.ylabel('Precission')

    plt.ylim(0, 100)

    x = np.arange(len(precission))
    rects = plt.bar(x, precission)
    plt.xticks(x, ('CG', 'BFGS', 'L-BFGS-B', 'TNC', 'SLSQP'))
    for rect in rects:
        height = rect.get_height()
        plt.annotate('{}%'.format(height),
                     xy=(rect.get_x() + rect.get_width() / 2,
                     height / 2),
                     xytext=(0, 0),  # 3 points vertical offset
                     textcoords="offset points",
                     ha='center', va='bottom', color=(0.0, 0.0, 0.0, 1.0),
                     fontsize=20, weight='bold')
    plt.show()

# Carga un fichero ".csv" y devuelve los datos
def load_data(file_name):
    values = read_csv(file_name, header=None).values
    return values.astype(float)

def normalize_matrix(X):
    mu = np.mean(X, axis=0)
    X_norm = X - mu

    sigma = np.std(X_norm, axis=0)
    X_norm = X_norm / sigma

    return X_norm


def get_data_matrix(data):
    X = np.delete(data, data.shape[1] - 1, axis=1) # (1200, 20)
    X = normalize_matrix(X)
```

```
    Y = data[:, data.shape[1] - 1] # (1200,)

    return X, Y

def main():
    train_data = load_data("../ProcessedDataSet/train.csv")
    validation_data = load_data("../ProcessedDataSet/validation.csv")
    test_data = load_data("../ProcessedDataSet/test.csv")

    X, Y = get_data_matrix(train_data)
    X_val, Y_val = get_data_matrix(validation_data)
    X_test, Y_test = get_data_matrix(test_data)

    input_layer = X.shape[1]
    hidden_layer = 16
    output_layer = 4

    cg_precission = neural_network.training_neural_network(X, Y, X_val, Y_val, X_test
, Y_test, input_layer, \
        hidden_layer, output_layer, 'CG', True)
    bfgs_precission = neural_network.training_neural_network(X, Y, X_val, Y_val, X_te
st, Y_test, input_layer, \
        hidden_layer, output_layer, 'BFGS', True)
    l_bfgs_b_precission = neural_network.training_neural_network(X, Y, X_val, Y_val,
X_test, Y_test, input_layer, \
        hidden_layer, output_layer, 'L-BFGS-B', True)
    tnc_precission = neural_network.training_neural_network(X, Y, X_val, Y_val, X_tes
t, Y_test, input_layer, \
        hidden_layer, output_layer, 'TNC', True)
    slsqp_precission = neural_network.training_neural_network(X, Y, X_val, Y_val, X_t
est, Y_test, input_layer, \
        hidden_layer, output_layer, 'SLSQP', True)

    precission = [cg_precission, bfgs_precission, l_bfgs_b_precission, tnc_precission
, slsqp_precission]
    draw_precission(precission)

main()
```

Código usado para Support Vector Machines

svm.py

```
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import svm
from sklearn.metrics import precision_score
import math
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
from sklearn.metrics import plot_confusion_matrix
from sklearn.feature_selection import SelectKBest, f_classif, chi2, mutual_info_class
if
```

```
def getSampleError(Xval, yval, Xtrain, ytrain, classificator, testingValue):
    '''
    Funcion para calcular el error dados unos ejemplos de entrenamiento, ejemplos de
validadion, un clasificador
    y unos valores a probar
    '''

    # calculamos el error con los ejemplos de validacion
    predictedValY = classificator.predict(Xval)
    validation_error = 100.0 * \
        float(sum(predictedValY != yval))/yval.shape[0]

    # calculamos el error con los ejemplos de entrenamiento
    predictedTrainY = classificator.predict(Xtrain)
    train_error = 100.0 * \
        float(sum(predictedTrainY != ytrain))/ytrain.shape[0]

   # print('{}\t{}\t{}\n'.format(testingValue, train_error, validation_error))

    return train_error, validation_error


def trainSigmoidSVM(X_train, y_train, Xval, yval, X_test, y_test):
     # Despues de haber calculado los errores con distintos grados, hemos visto que e
l resultado variaba en funcion
    # del valor de C, y no el grado, asi que calculamos directamente el mejor C

    C_test_values = [0.0001, 0.001, 0.01, 0.03, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.
8, 0.9, 1.0, 1.5, 2.0, 2.5, 3.0,
                     5.0, 7.0, 10.0, 15.0, 18.0, 21.0, 25.0, 27.0, 30.0, 35.0, 37.0,
40.0]

    error_train = []
    error_val = []

    for testing_c in C_test_values:

        svclassifier = svm.SVC(
            kernel='sigmoid', C=testing_c)
        svclassifier.fit(X_train, y_train)

        train_error, val_error = getSampleError(
            Xval, yval, X_train, y_train, svclassifier, testing_c)

        error_train.append(train_error)
        error_val.append(val_error)

        print(
            f'C{testing_c}:\t {train_error}\t {val_error}')
```

```
    # Vemos que el mejor valor es con C = 0.6


def trainGaussianSVM(X_train, y_train, Xval, yval, X_test, y_test):
    # Despues de haber calculado los errores con distintos grados, hemos visto que el
 resultado variaba en funcion
    # del valor de C, y no el grado, asi que calculamos directamente el mejor C

    C_test_values = [0.0001, 0.001, 0.01, 0.03, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0,
                     5.0, 7.0, 10.0, 15.0, 18.0, 21.0, 25.0, 27.0, 30.0, 35.0, 37.0,
40.0]

    error_train = []
    error_val = []

    print('degree and C \tTrain Error\tValidation Error\n')

    for testing_c in C_test_values:
        svclassifier = svm.SVC(
            kernel='rbf', C=float(testing_c))
        svclassifier.fit(X_train, y_train)

        train_error, val_error = getSampleError(
            Xval, yval, X_train, y_train, svclassifier, testing_c)

        error_train.append(train_error)
        error_val.append(val_error)

        print(f'C{testing_c}:\t {train_error}\t {val_error}')
    draw_findingBestValue(C_test_values, error_val, error_train,
                          'Finding Best C for gaussian kernel', 'C')


def draw_findingBestValue(test_values, error_val, error_train, title, xLabelText):
    plt.figure(figsize=(8, 5))
    plt.plot(test_values, error_val, 'or--', label='Validation Set Error')
    plt.plot(test_values, error_train, 'bo--', label='Training Set Error')
    plt.xlabel(f'${xLabelText}$ Value', fontsize=16)
    plt.ylabel('Classification Error [%]', fontsize=14)
    plt.title(title, fontsize=18)
    plt.legend()
    plt.show()


def trainPolynomialSVM(X_train, y_train, Xval, yval, X_test, y_test):
    test_degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
    error_train = []
    error_val = []
```

```
    print('Degree\tTrain Error\tValidation Error\n')
    for tDegree in test_degrees:
        svclassifier = svm.SVC(kernel='poly', degree=tDegree)
        svclassifier.fit(X_train, y_train)

        train_error, val_error = getSampleError(
            Xval, yval, X_train, y_train, svclassifier, tDegree)

        error_train.append(train_error)
        error_val.append(val_error)

    draw_findingBestValue(test_degrees, error_val, error_train,
                          'Finding Best polynomial degree value', 'Degree')
    # Parece que el mejor grado es un grado 3


def findBetterCForLinear(Xtrain, ytrain, Xval, yval):
    C_test_values = [0.0001, 0.001, 0.01, 0.03, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0,
                     5.0, 7.0, 10.0, 15.0, 18.0, 21.0, 25.0, 27.0, 30.0, 35.0, 37.0,
40.0]

    error_train = []
    error_val = []

    print('C\tTrain Error\tValidation Error\n')

    for testing_c in C_test_values:

        linear_svm = svm.SVC(C=testing_c, kernel='linear')

        # Ajustamos el kernel a los ejemplos de entrenamiento
        linear_svm.fit(Xtrain, ytrain)

        train_error, val_error = getSampleError(
            Xval, yval, Xtrain, ytrain, linear_svm, testing_c)

        error_train.append(train_error)
        error_val.append(val_error)

    #draw_findingBestValue(C_test_values, error_train,error_val, 'Finding Best C Valu
e', 'C')
    # De la grafica podemos observar que los mejores valores se encuentrar alrededor
de utilizar una C con un
    # valor de  37.0


def trainSVMForLinear(X_train, y_train, X_test, y_test, yval, Xval):
    findBetterCForLinear(X_train, y_train, Xval, yval)
```

```
def testLinearSVM(X_train, y_train, X_test, y_test):
    svclassifier = svm.SVC(kernel='linear', C=37.0)
    svclassifier.fit(X_train, y_train)

    # Ahora que hemos encontrado un buen valor de C, lo comprobamos contra los ejempl
os
    # de prueba
    y_pred = svclassifier.predict(X_test)
    linearSVMAccuracy = precision_score(y_test, y_pred, average='micro')*100.

    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    print(
        f"La precisión del kernel lineal con un valor C = 37.0 es del {linearSVMAccur
acy}%")

    plotConfusionMatrix(svclassifier, X_test,
                        y_test, 'true', "Linear")

    return linearSVMAccuracy, svclassifier


def testSigmoidSVM(X_train, y_train, X_test, y_test):
    bestC = 0.6
    finalSigmoidClassifier = svm.SVC(kernel='sigmoid', C=bestC)
    finalSigmoidClassifier.fit(X_train, y_train)

    y_pred = finalSigmoidClassifier.predict(X_test)
    sigmoidSVMAccuracy = precision_score(y_test, y_pred, average='micro')*100.

    print(
        f"La precisión del kernel sigmoide con un valor de C = {bestC} es del {sigmoi
dSVMAccuracy}%")

    plotConfusionMatrix(finalSigmoidClassifier, X_test,
                        y_test, 'true', "Sigmoid")
    return sigmoidSVMAccuracy, finalSigmoidClassifier


def testGaussianSVM(X_train, y_train, X_test, y_test):
    bestC = 2.8
    finalGaussianClassifier = svm.SVC(kernel='rbf', C=bestC)
    finalGaussianClassifier.fit(X_train, y_train)

    y_pred = finalGaussianClassifier.predict(X_test)
    gaussianSVMAccuracy = precision_score(y_test, y_pred, average='micro')*100.
```

```
    print(
        f"La precisión del kernel gaussiano con un valor de C = {bestC} es del {gauss
ianSVMAccuracy}%")

    plotConfusionMatrix(finalGaussianClassifier, X_test,
                        y_test, 'true', "Gaussian")

    return gaussianSVMAccuracy, finalGaussianClassifier


def testPolynomialSVM(X_train, y_train, X_test, y_test):
    finalPolyClassifier = svm.SVC(kernel='poly', degree=3)
    finalPolyClassifier.fit(X_train, y_train)

    y_pred = finalPolyClassifier.predict(X_test)
    polySVMAccuracy = precision_score(y_test, y_pred, average='micro')*100.

    print(
        f"La precisión del kernel polinómico con un grado = 3 es del {polySVMAccuracy
}%")

    plotConfusionMatrix(finalPolyClassifier, X_test,
                        y_test, 'true', "Polynomial")
    return polySVMAccuracy, finalPolyClassifier


def show_global_accuracy(X_train, y_train, X_test, y_test):
    preccision = [testSigmoidSVM(X_train, y_train, X_test, y_test), testGaussianSVM(X
_train, y_train, X_test, y_test),
                  testPolynomialSVM(X_train, y_train, X_test, y_test),  testLinearSVM
(X_train, y_train, X_test, y_test)]

    draw_different_kernels_accuracy(preccision)


def draw_different_kernels_accuracy(precission):
    plt.figure(figsize=(30, 20))
    plt.title('SVM kernels accuracy ')
    plt.xlabel('Kernel')
    plt.ylabel('Precission[%]')

    plt.ylim(0, 110)

    x = np.arange(len(precission))
    rects = plt.bar(x, precission, color='red')
    plt.xticks(x, ('Sigmoid', 'Gaussian', 'Polynomial', 'Linear',))
    for rect in rects:
        height = rect.get_height()
```

```
        plt.annotate('{}%'.format(height),
                     xy=(rect.get_x() + rect.get_width() / 2, height),
                     xytext=(0, 3),  # 3 points vertical offset
                     textcoords="offset points",
                     ha='center', va='bottom')
    plt.show()


def plotConfusionMatrix(classifier, X_test, y_test, normalize, classifierName):
    disp = plot_confusion_matrix(classifier, X_test, y_test,
                                 display_labels=[
                                     'low cost', 'medium cost', 'high cost', 'very hi
gh cost'],
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(f'Normalized confusion matrix for {classifierName}')

    plt.show()


'''
    Univariate feature selection works by selecting the best features based on
    univariate statistical tests. It can be seen as a preprocessing step to an estima
tor.
    Scikit-
learn exposes feature selection routines as objects that implement the transform meth
od:

    Para clasificacion se puede usar:
       chi2, f_classif, mutual_info_classif
'''


def univariate_feature_selection(X_train, y_train):

    selector = SelectKBest(f_classif,
                           k=2).fit_transform(X_train, y_train)

    return selector


def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy
```

```
def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


def draw_sets(X_train, y_train, classifier, name):

    X = univariate_feature_selection(X_train, y_train)
    clf = classifier.fit(X, y_train)
    fig, ax = plt.subplots()
    title = (f'Decision surface for kernel {name}')

    X0, X1 = X[:, 0], X[:, 1]
    xx, yy = make_meshgrid(X0, X1)

    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_ylabel('Mobile spectrum')
    ax.set_xlabel('2 Most significant attributes using f_classif method')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    ax.legend()
    plt.show()
```

main.py

```
import numpy as np
from pandas.io.parsers import read_csv
import svm as ourSVM


def load_data(file_name):
    values = read_csv(file_name, header=None).values
    return values.astype(float)


def normalize_matrix(X):
    mu = np.mean(X, axis=0)
    X_norm = X - mu

    sigma = np.std(X_norm, axis=0)
    X_norm = X_norm / sigma

    return X_norm
```

```
def get_data_matrix(data):
    X = np.delete(data, data.shape[1] - 1, axis=1)
    X = normalize_matrix(X)
    X = np.insert(X, 0, 1, axis=1)
    Y = data[:, data.shape[1] - 1]

    return X, Y


def main():
    train_data = load_data("../ProcessedDataSet/train.csv")
    validation_data = load_data("../ProcessedDataSet/validation.csv")
    test_data = load_data("../ProcessedDataSet/test.csv")

    X, Y = get_data_matrix(train_data)
    X_val, Y_val = get_data_matrix(validation_data)
    X_test, Y_test = get_data_matrix(test_data)

    '''Training Support Vector Machines '''
    #ourSVM.trainSVMForLinear(X, Y, X_test, Y_test, Y_val, X_val)
    #ourSVM.trainPolynomialSVM(X, Y, X_val, Y_val, X_test, Y_test)
    #ourSVM.trainGaussianSVM(X, Y, X_val, Y_val, X_test, Y_test)
    #ourSVM.trainSigmoidSVM(X, Y, X_val, Y_val, X_test, Y_test)

    '''Testing SVM'''
    sigmoidClassifier = ourSVM.testSigmoidSVM(X, Y, X_test, Y_test)[1]
    gaussianClassifier = ourSVM.testGaussianSVM(X, Y, X_test, Y_test)[1]
    polynomialClassifier = ourSVM.testPolynomialSVM(X, Y, X_test, Y_test)[1]
    linearClassifier = ourSVM.testLinearSVM(X, Y, X_test, Y_test)[1]

    classifiers = [sigmoidClassifier, gaussianClassifier,
                   polynomialClassifier, linearClassifier]

    names = ["Sigmoid", "Gaussian", "Polynomial", "Linear"]
    # Draw global accuracy plot
    #ourSVM.show_global_accuracy(X, Y, X_test, Y_test)

    for i in range(len(classifiers)):
        ourSVM.draw_sets(X_test, Y_test, classifiers[i], names[i])


main()
```

Código usado para Keras

regresion.py

```
from pandas.io.parsers import read_csv
import numpy as np
import matplotlib.pyplot as plt

from keras import optimizers
from keras.models import Sequential
from keras.layers.core import Dense
import tensorflow as tf


def load_data(file_name):
    values = read_csv(file_name, header=None).values
    return values.astype(float)


def normalize_matrix(X):
    mu = np.mean(X, axis=0)
    X_norm = X - mu

    sigma = np.std(X_norm, axis=0)
    X_norm = X_norm / sigma

    return X_norm


def get_data_matrix(data):
    X = np.delete(data, data.shape[1] - 1, axis=1)  # (1200, 20)
    X = normalize_matrix(X)
    X = np.insert(X, 0, 1, axis=1)  # (1200, 21)
    Y = data[:, data.shape[1] - 1]  # (1200,)

    return X, Y

def main():
    train_data = load_data("../ProcessedDataSet/train.csv")
    X, Y = get_data_matrix(train_data)

    lambda_vec = [0, 0.001, 0.003, 0.01, 0.03]

    for i in range(len(lambda_vec)):
        model = Sequential()
        model.add(Dense(units=4, input_shape=(21,), activation='sigmoid'))

        model.compile(optimizer=optimizers.Adam(lr=lambda_vec[i]), loss='sparse_categ
orical_crossentropy', metrics=['accuracy'])

        # Plot training & validation accuracy values
        history = model.fit(x=X, y=Y, validation_split=0.25, batch_size=16, verbose=1
, epochs=100)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('Model accuracy with $\lambda$ = {}'.format(lambda_vec[i]))
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

        # Plot training & validation loss values
```

```
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('Model loss with $\lambda$ = {}'.format(lambda_vec[i]))
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

main()
```

red.py

```
from pandas.io.parsers import read_csv
import numpy as np
import matplotlib.pyplot as plt

from keras import optimizers
from keras.models import Sequential
from keras.layers.core import Dense
import tensorflow as tf


def load_data(file_name):
    values = read_csv(file_name, header=None).values
    return values.astype(float)


def normalize_matrix(X):
    mu = np.mean(X, axis=0)
    X_norm = X - mu

    sigma = np.std(X_norm, axis=0)
    X_norm = X_norm / sigma

    return X_norm


def get_data_matrix(data):
    X = np.delete(data, data.shape[1] - 1, axis=1)  # (1200, 20)
    X = normalize_matrix(X)
    X = np.insert(X, 0, 1, axis=1)  # (1200, 21)
    Y = data[:, data.shape[1] - 1]  # (1200,)

    return X, Y

def main():
    train_data = load_data("../ProcessedDataSet/train.csv")
    X, Y = get_data_matrix(train_data)

    lambda_vec = [0, 0.001, 0.003, 0.01, 0.03]

    for i in range(len(lambda_vec)):
        model = Sequential()
        model.add(Dense(units=X.shape[1], input_shape=(X.shape[1],), activation='tanh
'))
        model.add(Dense(units=8, activation='tanh'))
        model.add(Dense(units=4, activation='sigmoid'))
```

```
        model.compile(optimizer=optimizers.Adam(lr=lambda_vec[i]), loss='sparse_categ
orical_crossentropy', metrics=['accuracy'])

        # Plot training & validation accuracy values
        history = model.fit(x=X, y=Y, validation_split=0.25, batch_size=16, verbose=1
, epochs=100)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('Model accuracy with $\lambda$ = {}'.format(lambda_vec[i]))
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

        # Plot training & validation loss values
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('Model loss with $\lambda$ = {}'.format(lambda_vec[i]))
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()


main()
```

Código usado para Phone CompluHouse

run.py

```
import numpy as np
from flask import Flask, render_template, request, jsonify
import mobile_price

app = Flask(__name__, static_url_path='')

num = 0

@app.route('/')
@app.route('/index.html')
def index():
    return render_template('index.html')

@app.route('/main', methods=['GET', 'POST'])
def main():

    bateria_val = request.args.get('bateria_val', 0, type=float)
    blue_val = request.args.get('blue_val', 0, type=float)
    procesador_val = request.args.get('procesador_val', 0, type=float)
    sim_val = request.args.get('sim_val', 0, type=float)
    cam_fron_val = request.args.get('cam_fron_val', 0, type=float)
    cuatrog_val = request.args.get('cuatrog_val', 0, type=float)
    memoria_val = request.args.get('memoria_val', 0, type=float)
    profundidad_val = request.args.get('profundidad_val', 0, type=float)
    peso_val = request.args.get('peso_val', 0, type=float)
    nucleos_val = request.args.get('nucleos_val', 0, type=float)
```

```
    cam_prin_val = request.args.get('cam_prin_val', 0, type=float)
    res_alto_val = request.args.get('res_alto_val', 0, type=float)
    res_ancho_val = request.args.get('res_ancho_val', 0, type=float)
    ram_val = request.args.get('ram_val', 0, type=float)
    altura_val = request.args.get('altura_val', 0, type=float)
    anchura_val = request.args.get('anchura_val', 0, type=float)
    autonomia_val = request.args.get('autonomia_val', 0, type=float)
    tresg_val = request.args.get('tresg_val', 0, type=float)
    tactil_val = request.args.get('tactil_val', 0, type=float)
    wifi_val = request.args.get('wifi_val', 0, type=float)

    file_num = request.args.get('file_num', 0, type=int)

    a = np.array([[
        bateria_val,
        blue_val,
        procesador_val,
        sim_val,
        cam_fron_val,
        cuatrog_val,
        memoria_val,
        profundidad_val,
        peso_val,
        nucleos_val,
        cam_prin_val,
        res_alto_val,
        res_ancho_val,
        ram_val,
        altura_val,
        anchura_val,
        autonomia_val,
        tresg_val,
        tactil_val,
        wifi_val
        ]])
    np.savetxt('../ProcessedDataSet/user.csv', (a), delimiter=',')

    res = mobile_price.calculate_range_price()

    f= open("static/result" + str(file_num) + ".txt", "w+")

    if res == 0:
        f.write("GAMA BAJA")

    elif res == 1:
        f.write("GAMA MEDIA")

    elif res == 2:
        f.write("GAMA ALTA")

    elif res == 3:
        f.write("GAMA SUPERIOR")

    f.close

    return jsonify(result=0)
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

mobile_price.py

```
import numpy as np
from pandas.io.parsers import read_csv
from scipy.io import loadmat

def load_data(file_name):
    values = read_csv(file_name, header=None).values
    return values.astype(float)

# Función sigmoide
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def normalize_matrix(X):
    mu = np.mean(X, axis=0)
    X_norm = X - mu

    sigma = np.std(X_norm, axis=0)
    X_norm = X_norm / sigma

    return X_norm


def get_data_matrix(data, X_toAdd):
    X = np.delete(data, data.shape[1] - 1, axis=1) # (1200, 20)
    X = np.vstack((X, X_toAdd))
    X = normalize_matrix(X)

    return X

# Devuelve "Y" a partir de una X y no unos pesos determinados
def forward_propagate(X, theta1, theta2):
    m = X.shape[0]

    a1 = np.hstack([np.ones([m, 1]), X])    # (5000, 401)
    z2 = np.dot(a1, theta1.T)   # (5000, 25)

    a2 = np.hstack([np.ones([m, 1]), sigmoid(z2)])  # (5000, 26)
    z3 = np.dot(a2, theta2.T)   # (5000, 10)

    h = sigmoid(z3) # (5000, 10)

    return a1, z2, a2, z3, h

def calculate_range_price():

    values = read_csv("../ProcessedDataSet/user.csv", header=None).values
    X_user = values.astype(float)

    data = load_data("../ProcessedDataSet/test.csv")
    X = get_data_matrix(data, X_user)

    weights = loadmat("weights.mat")
```

```
    theta1 = weights["Theta1"] # (8 x 21)
    theta2 = weights["Theta2"] # (4 x 9)

    a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)

    res = np.argmax(h[h.shape[0] - 1])
    return res
```

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Phone CompluHouse</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <link href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css" rel="stylesheet">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></
script>
    <script>
        function rangeSlider(value, name){
            document.getElementsByName(name)[0].innerHTML = value;
        }
    </script>
    <script>
        function initStorage(){
            if (typeof(Storage) !== "undefined") {
                if(typeof(localStorage.getItem("file_num")) == undefined){
                    localStorage.setItem("file_num", "0");
                }
            }
        }

        function goPython(){
            ajax1();
            ajax2();

        }

        function ajax1(){
            var frm = $("#id_form");
            var datos = frm.serialize();

            if (typeof(Storage) !== "undefined") {
                localStorage.setItem("file_num", Number(localStorage.getItem("file_nu
m")) + 1);
                console.log(localStorage.getItem("file_num"));
            }
            else{
                console.log("Po no existe");
            }
            $.ajax({
                url: '/main',
                method: frm.prop('method'),
                timeout: 5000,
                async: false,
                data:{
```

```
                    'bateria_val': $("#bateria_val").val(),
                    'blue_val': $("#blue_val").val(),
                    'procesador_val': $("#procesador_val").val(),
                    'sim_val': $("#sim_val").val(),
                    'cam_fron_val': $("#cam_fron_val").val(),
                    'cuatrog_val': $("#cuatrog_val").val(),
                    'memoria_val': $("#memoria_val").val(),
                    'profundidad_val': $("#profundidad_val").val(),
                    'peso_val': $("#peso_val").val(),
                    'nucleos_val': $("#nucleos_val").val(),
                    'cam_prin_val': $("#cam_prin_val").val(),
                    'res_alto_val': $("#res_alto_val").val(),
                    'res_ancho_val': $("#res_ancho_val").val(),
                    'ram_val': $("#ram_val").val(),
                    'altura_val': $("#altura_val").val(),
                    'anchura_val': $("#anchura_val").val(),
                    'autonomia_val': $("#autonomia_val").val(),
                    'tresg_val': $("#tresg_val").val(),
                    'tactil_val': $("#tactil_val").val(),
                    'wifi_val': $("#wifi_val").val(),
                    'file_num': localStorage.getItem("file_num")
                },
                dataType: 'JSON',
                success : function(e) {
                    console.log("Success");
                },
                error : function(e) {
                    console.info("Error");
                },
                done : function(e) {
                    console.info("DONE");
                }
            });
        }

        function ajax2(){
            $.ajax({
                url: "result" + localStorage.getItem("file_num") + ".txt",
                dataType: "text",
                success: function(result, response){
                    alert("El teléfono móvil que has elegido es de " + result);
                },
                error: function(e){
                    $.ajax(this);
                }
            });
        }

    </script>
</head>
<body onload="initStorage()">
    <div class="content">
        <div class="title">
            <h1 id="div1">PHONE COMPLUHOUSE</h1>
        </div>
        <div class="filter_user">
            <form id="id_form" method="get">
                <div class="filter">
```

```
                    <div class=range_box>
                        <a>BATERÍA</a>
                        <input id="bateria_val" type="range" class="range" name=""
                        value="0" min="0" max="2000"
                        onmousemove="rangeSlider(this.value, 'bateria_range')"
                        onChange="rangeSlider(this.value, 'bateria_range')">
                        <span id="rangeValue" name="bateria_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <a>BLUETOOTH</a>
                    <select id="blue_val" required>
                        <option value="">Seleccionar ...</option>
                        <option value="0">No</option>
                        <option value="1">Sí</option>
                    </select>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>VELOCIDAD DEL PROCESADOR</a>
                        <input id="procesador_val" type="range" class="range" name=""

                        value="0" min="0" max="3" step="0.1"
                        onmousemove="rangeSlider(this.value, 'procesador_range')"
                        onChange="rangeSlider(this.value, 'procesador_range')">
                        <span id="rangeValue" name="procesador_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <a>DUAL SIM</a>
                    <select id="sim_val" required>
                        <option value="">Seleccionar ...</option>
                        <option value="0">No</option>
                        <option value="1">Sí</option>
                    </select>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>CÁMARA FRONTAL, MP</a>
                        <input id="cam_fron_val" type="range" class="range" name=""
                        value="0" min="0" max="20"
                        onmousemove="rangeSlider(this.value, 'cam_frontal_range')"
                        onChange="rangeSlider(this.value, 'cam_frontal_range')">
                        <span id="rangeValue" name="cam_frontal_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <a>4G</a>
                    <select id="cuatrog_val" required>
                        <option value="">Seleccionar ...</option>
                        <option value="0">No</option>
                        <option value="1">Sí</option>
                    </select>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>MEMORIA INTERNA</a>
                        <input id="memoria_val" type="range" class="range" name=""
```

```
                        value="0" min="0" max="64"
                        onmousemove="rangeSlider(this.value, 'memoria_range')"
                        onChange="rangeSlider(this.value, 'memoria_range')">
                        <span id="rangeValue" name="memoria_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>PROFUNDIDAD DEL TELÉFONO, CM</a>
                        <input id="profundidad_val" type="range" class="range" name="
"
                        value="0" min="0" max="1" step="0.1"
                        onmousemove="rangeSlider(this.value, 'profundidad_range')"
                        onChange="rangeSlider(this.value, 'profundidad_range')">
                        <span id="rangeValue" name="profundidad_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>PESO</a>
                        <input id="peso_val" type="range" class="range" name=""
                        value="0" min="80" max="200"
                        onmousemove="rangeSlider(this.value, 'peso_range')"
                        onChange="rangeSlider(this.value, 'peso_range')">
                        <span id="rangeValue" name="peso_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>NÚMERO DE NÚCLEOS DEL PROCESADOR</a>
                        <input id="nucleos_val" type="range" class="range" name=""
                        value="0" min="1" max="8"
                        onmousemove="rangeSlider(this.value, 'nucleos_range')"
                        onChange="rangeSlider(this.value, 'nucleos_range')">
                        <span id="rangeValue" name="nucleos_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>CÁMARA PRINCIPAL, MP</a>
                        <input id="cam_prin_val" type="range" class="range" name=""
                        value="0" min="0" max="20"
                        onmousemove="rangeSlider(this.value, 'cam_prin_range')"
                        onChange="rangeSlider(this.value, 'cam_prin_range')">
                        <span id="rangeValue" name="cam_prin_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>RESOLUCIÓN DEL ALTO DE LA PANTALLA</a>
                        <input id="res_alto_val" type="range" class="range" name=""
                        value="0" min="1" max="1980"
                        onmousemove="rangeSlider(this.value, 'res_alto_range')"
                        onChange="rangeSlider(this.value, 'res_alto_range')">
                        <span id="rangeValue" name="res_alto_range">0</span>
                    </div>
                </div>
                <div class="filter">
```

```
                    <div class=range_box>
                        <a>RESOLUCIÓN DEL ANCHO DE LA PANTALLA</a>
                        <input id="res_ancho_val" type="range" class="range" name=""
                        value="0" min="1" max="1980"
                        onmousemove="rangeSlider(this.value, 'res_ancho_range')"
                        onChange="rangeSlider(this.value, 'res_ancho_range')">
                        <span id="rangeValue" name="res_ancho_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>RAM</a>
                        <input id="ram_val" type="range" class="range" name=""
                        value="0" min="0" max="4000"
                        onmousemove="rangeSlider(this.value, 'ram_range')"
                        onChange="rangeSlider(this.value, 'ram_range')">
                        <span id="rangeValue" name="ram_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>ALTURA, CM</a>
                        <input id="altura_val" type="range" class="range" name=""
                        value="0" min="0" max="20"
                        onmousemove="rangeSlider(this.value, 'altura_range')"
                        onChange="rangeSlider(this.value, 'altura_range')">
                        <span id="rangeValue" name="altura_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>ANCHURA, CM</a>
                        <input id="anchura_val" type="range" class="range" name=""
                        value="0" min="0" max="20"
                        onmousemove="rangeSlider(this.value, 'anchura_range')"
                        onChange="rangeSlider(this.value, 'anchura_range')">
                        <span id="rangeValue" name="anchura_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <div class=range_box>
                        <a>AUTONOMÍA</a>
                        <input id="autonomia_val" type="range" class="range" name=""
                        value="0" min="0" max="20"
                        onmousemove="rangeSlider(this.value, 'autonomia_range')"
                        onChange="rangeSlider(this.value, 'autonomia_range')">
                        <span id="rangeValue" name="autonomia_range">0</span>
                    </div>
                </div>
                <div class="filter">
                    <a>3G</a>
                    <select id="tresg_val" required>
                        <option value="">Seleccionar ...</option>
                        <option value="0">No</option>
                        <option value="1">Sí</option>
                    </select>
                </div>
                <div class="filter">
```

```
                    <a>PANTALLA TÁCTIL</a>
                    <select id="tactil_val" required>
                        <option value="">Seleccionar ...</option>
                        <option value="0">No</option>
                        <option value="1">Sí</option>
                    </select>
                </div>
                <div class="filter">
                    <a>WIFI</a>
                    <select id="wifi_val" required>
                        <option value="">Seleccionar ...</option>
                        <option value="0">No</option>
                        <option value="1">Sí</option>
                    </select>
                </div>
                <input class="submit_button" type="button" onclick="goPython()" value
="Aplicar flitros">
            </form>
        </div>
    </div>
</body>
</html>
```

style.css

```
body{
    position: absolute;
    width: 100%;
    margin: 0;
    padding: 0;
    background: rgb(44, 44, 44);
    font-family: sans-serif;
}

.title{
    position: relative;
    margin: 0;
    margin-bottom: 20px;
    padding: 10px;
    width: 100%;
    background-image: linear-
gradient(to top, rgba(0, 0, 0, 0), rgb(33, 137, 255), rgb(33, 137, 255));
    color: rgb(44, 44, 44);
    font-size: 24px;
    font-weight: bold;
    text-align: center;
}

.filter{
    position: relative;
    width: 100%;
    margin: 0;
    padding: 30px;
    border-bottom: solid 1px;
    border-image-source:
        linear-gradient(
            45deg,
```

```
            rgb(148, 148, 148),
            rgba(0, 0, 0, 0),
            rgba(0, 0, 0, 0)
        );
    border-image-slice: 1;
}

.filter:nth-child(20){
    border: none;
}

.filter a{
    font-size: 22px;
    font-weight: bold;
    color: #eee;
}

.filter select{
    width: 250px;
    height: 50px;
    font-size: 20px;
    margin-left: 20px;
    background: #0563af;
    color: #fff;
    padding: 10px;
    border: none;
    box-shadow: 0 5px 25px rgba(0, 0, 0, 0.5);
    outline: none;
    border-radius: 5px;
}

.submit_button{
    position: relative;
    left: 50%;
    transform: translate(-50%, 0);
    margin: 50px;
    cursor: pointer;
    width: 300px;
    height: 80px;
    font-size: 36px;
    color: #0563af;
    font-weight: bold;
    border: 3px #0563af solid;
    background: #fff;

    border-radius: 20px;
    box-shadow: 0 5px 25px rgba(0, 0, 0, 0.5);
}

.submit_button:hover{
    color: rgb(44, 44, 44);
    border: 3px rgb(44, 44, 44) solid;
    background: #fff;
}

.range_box{
    position: relative;
    display: flex;
```

```
    font-size: 20px;
}

.range{
    position: relative;
    width: 300px;
    height: 2px;
    margin-top: 13px;
    margin-left: 30px;
    outline: none;
}

.range::-webkit-slider-runnable-track{
    background: #111;
    height: 2px;
}

.range::-webkit-slider-thumb{
    cursor: pointer;
    -webkit-appearance: none;
    box-sizing: content-box;
    transform: translateY(-10px);
    background: #0563af;
}

span{
    position: relative;
    top: 0px;
    left: 0;
    margin-left: 10px;
    padding: 2px;
    width: 50px;
    box-sizing: border-box;
    background: #fff;
    text-align: center;
    border-radius: 2px;
}

span:before{
    content: '';
    position: absolute;
    top: 50%;
    transform: translateY(-50%) rotate(45deg);
    left: -5px;
    width: 10px;
    height: 10px;
    background: #fff;
}
```

Apéndice B. Gráficas

Regresión Logística. Error de clasificación según el valor de *lambda.*

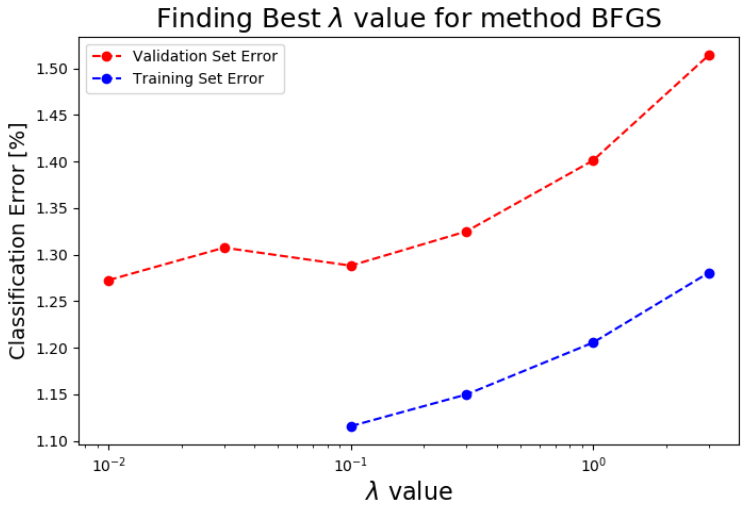


Ilustración 1. Error de clasificación para los distintos valores de lambda usando el algoritmo BFGS

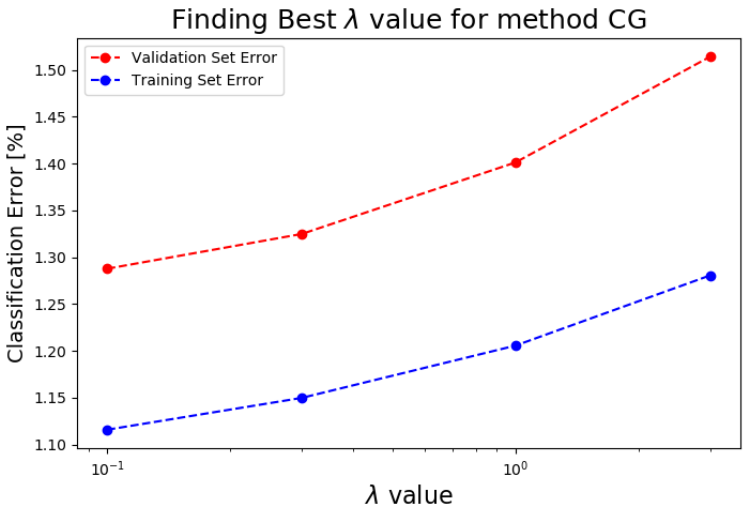


Ilustración 2. Error de clasificación para los distintos valores de lambda usando el algoritmo CG

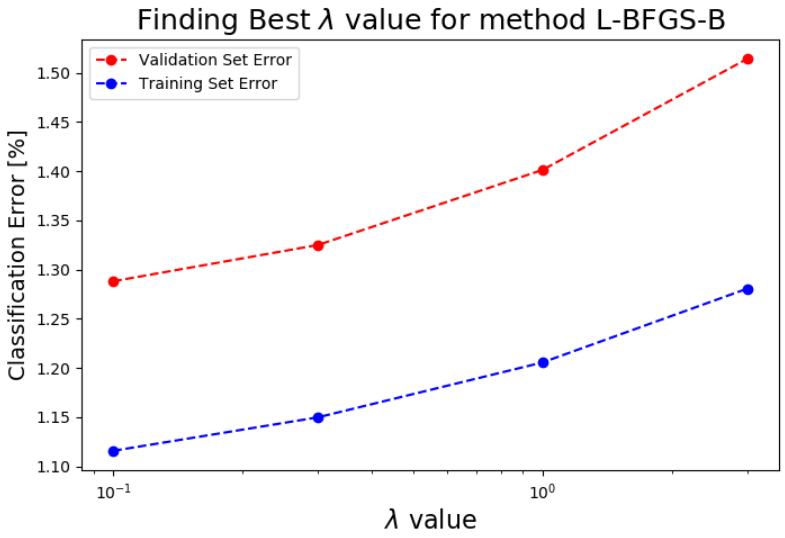


Ilustración 3. Error de clasificación para los distintos valores de lambda usando el algoritmo L-BFGS-B

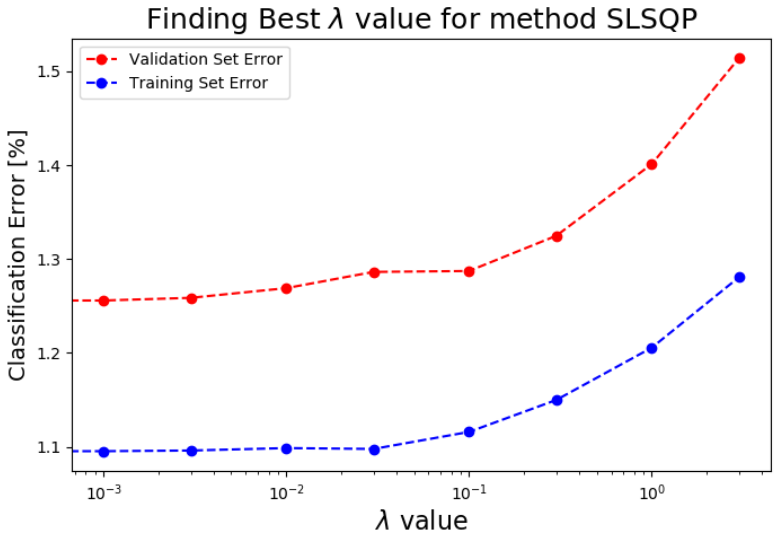


Ilustración 4. Error de clasificación para los distintos valores de lambda usando el algoritmo SLSQP

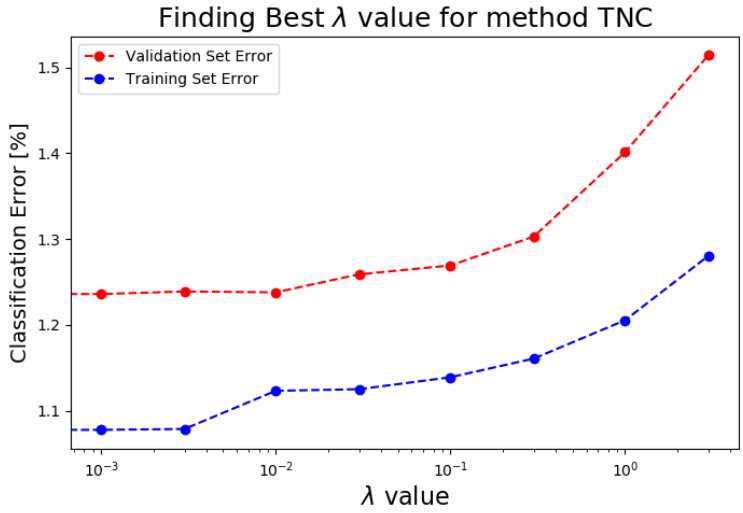


Ilustración 5. Error de clasificación para los distintos valores de lambda usando el algoritmo TNC

Regresión Logística. Precisión de los algoritmos con cada una de las *lambdas*.

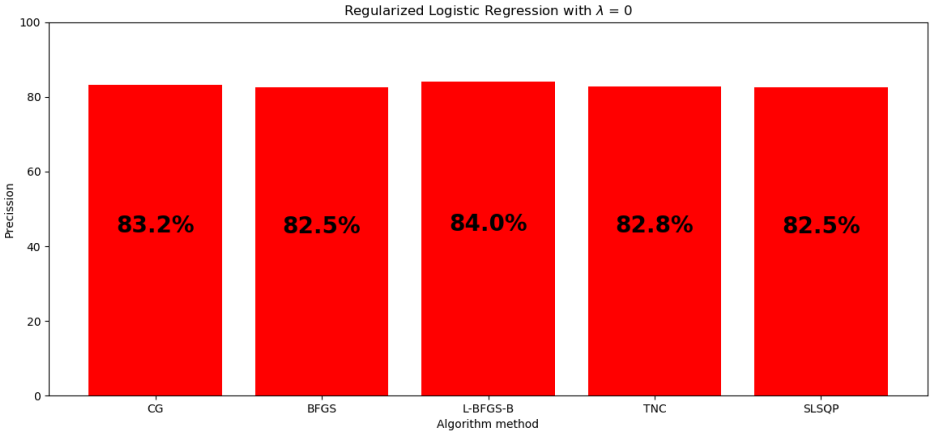


Ilustración 6. Precisión de la regresión logística regularizada para lambda 0

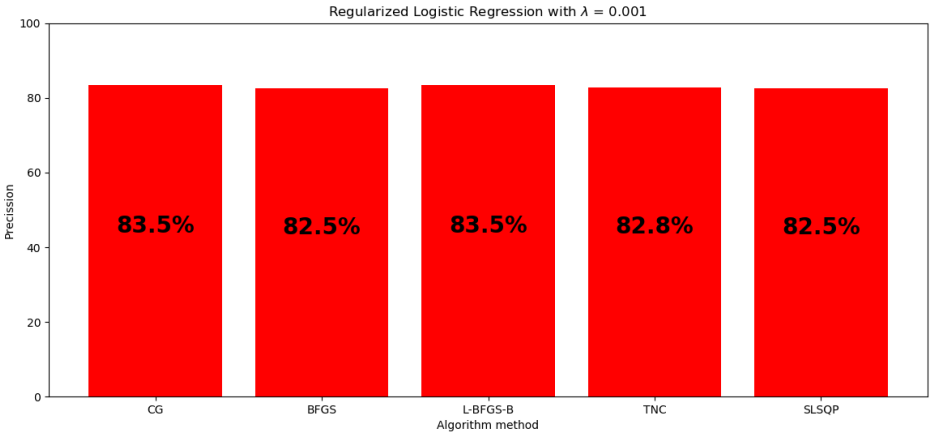


Ilustración 7. Precisión de la regresión logística regularizada para lambda 0.001

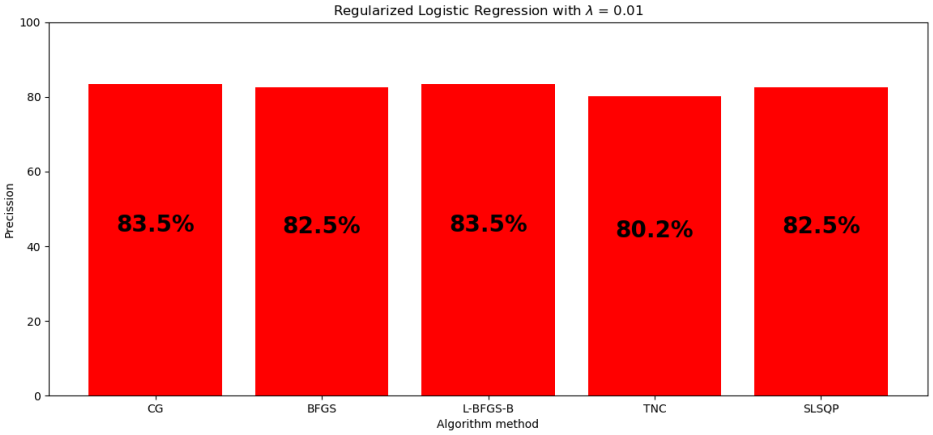


Ilustración 8. Precisión de la regresión logística regularizada para lambda 0.01

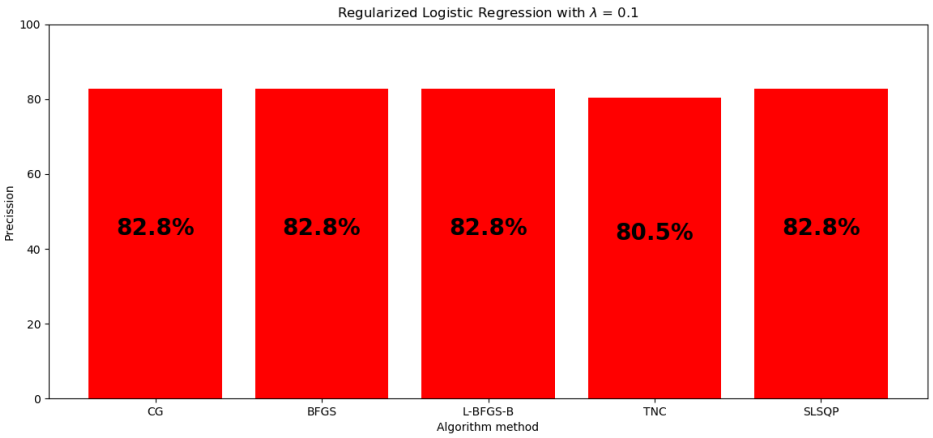


Ilustración 9. Precisión de la regresión logística regularizada para lambda 0.1

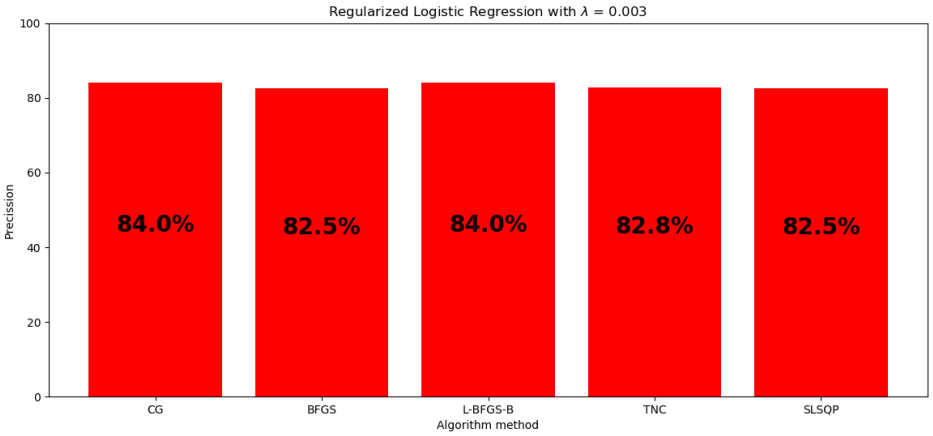


Ilustración 10. Precisión de la regresión logística regularizada para lambda 0.003

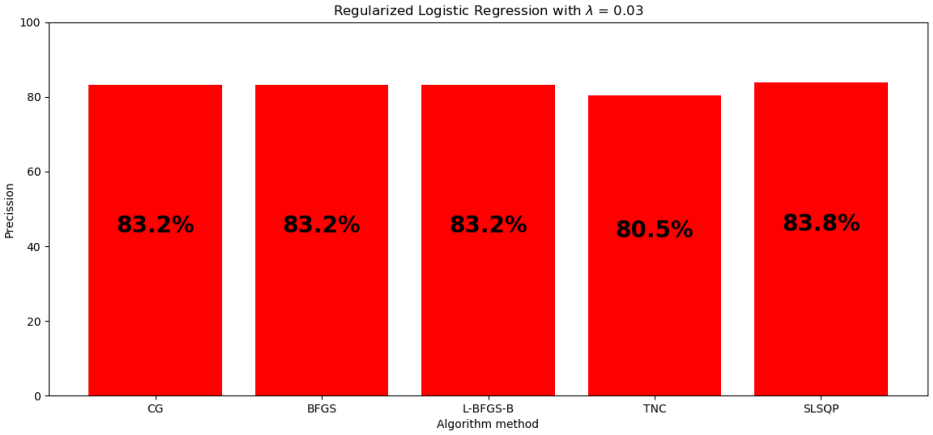


Ilustración 11. Precisión de la regresión logística regularizada para lambda 0.03

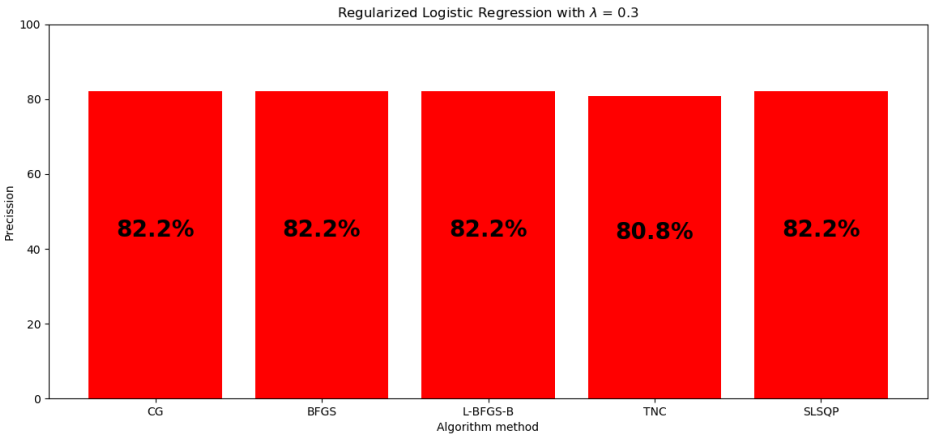


Ilustración 12. Precisión de la regresión logística regularizada para lambda 0.3

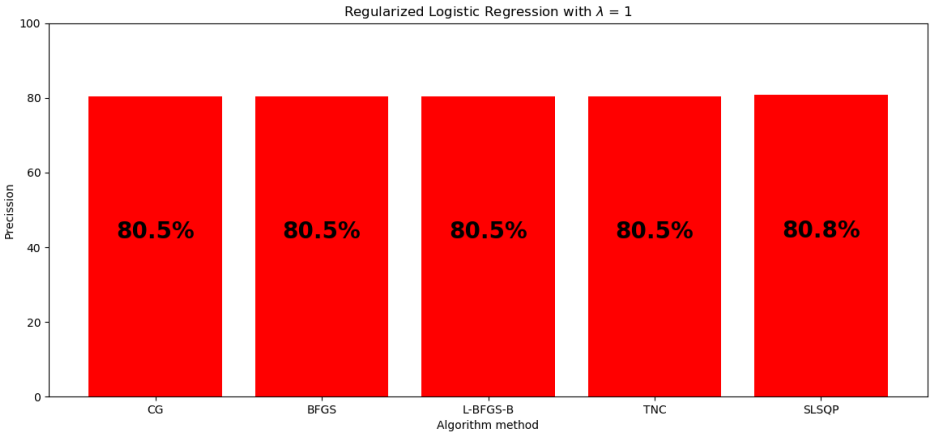


Ilustración 13. Precisión de la regresión logística regularizada para lambda 1

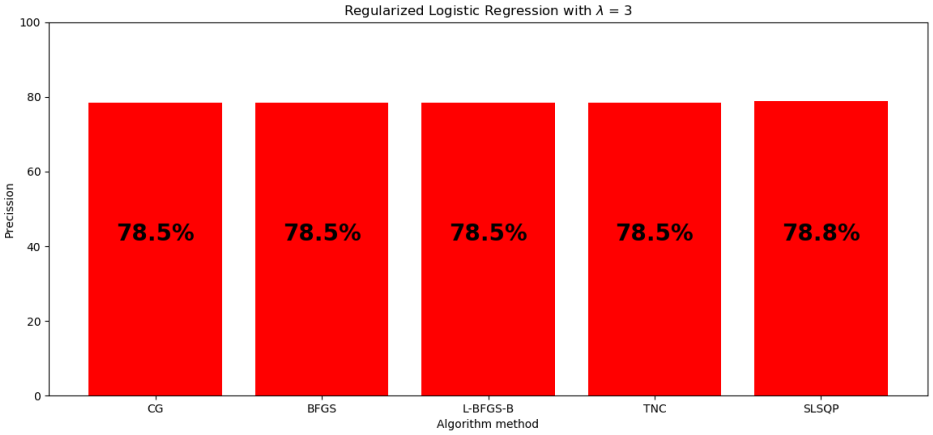


Ilustración 14. Precisión de la regresión logística regularizada para lambda 3

Regresión Logística. Precisión de los algoritmos con el mejor valor de *lambda* para cada uno de ellos.

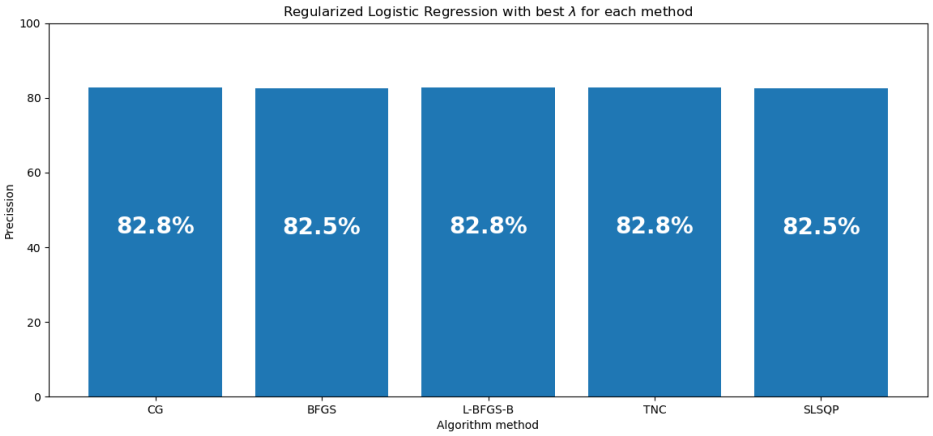


Ilustración 15. Precisión de la regresión logística regularizada con cada algoritmo

Redes Neuronales. Error de clasificación según el valor de *lambda.*

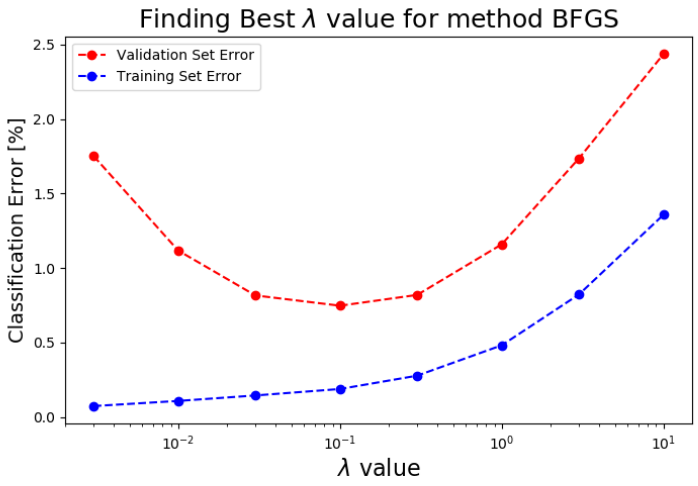


Ilustración 16. Error de clasificación para los distintos valores de lambda usando el algoritmo BFGS

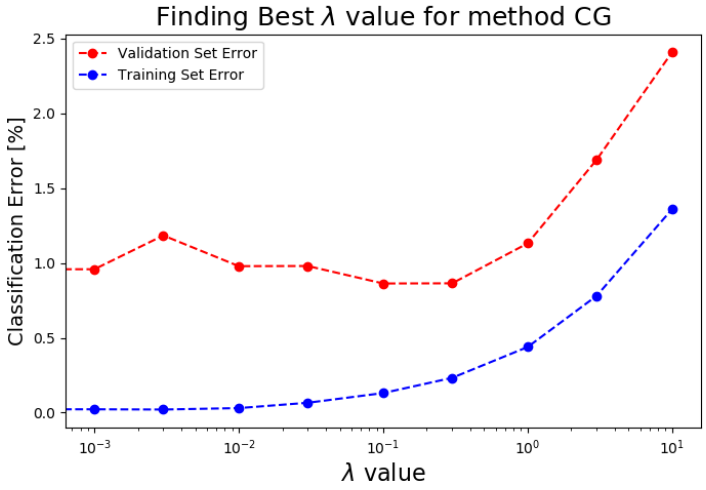


Ilustración 17. Error de clasificación para los distintos valores de lambda usando el algoritmo CG

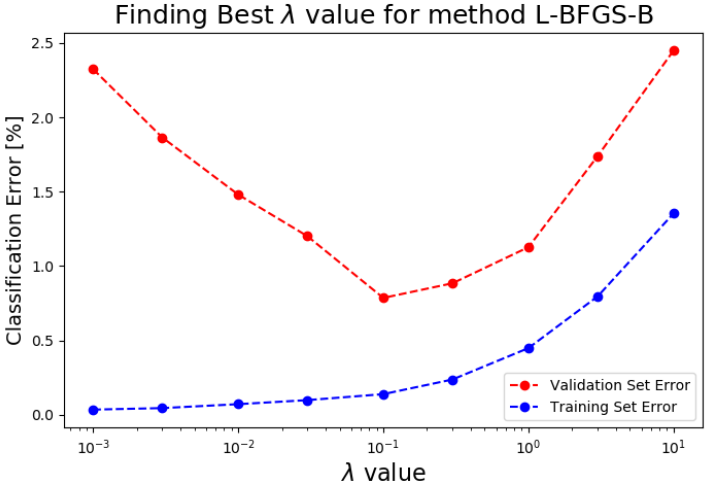


Ilustración 18. Error de clasificación para los distintos valores de lambda usando el algoritmo L-BFGS-B

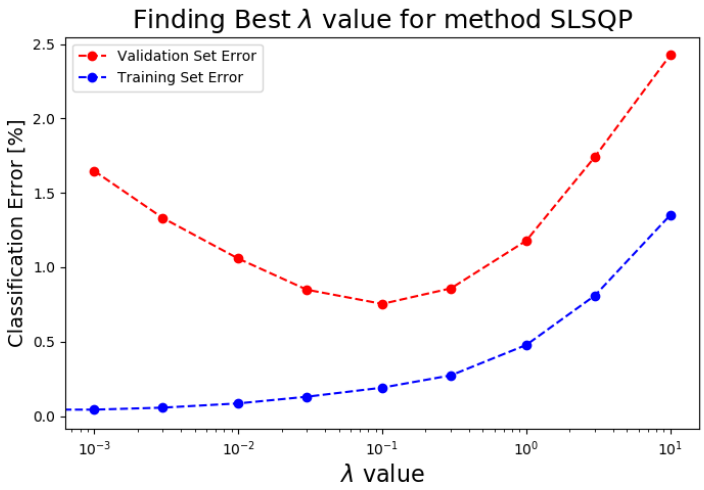


Ilustración 19. Error de clasificación para los distintos valores de lambda usando el algoritmo SLSQP

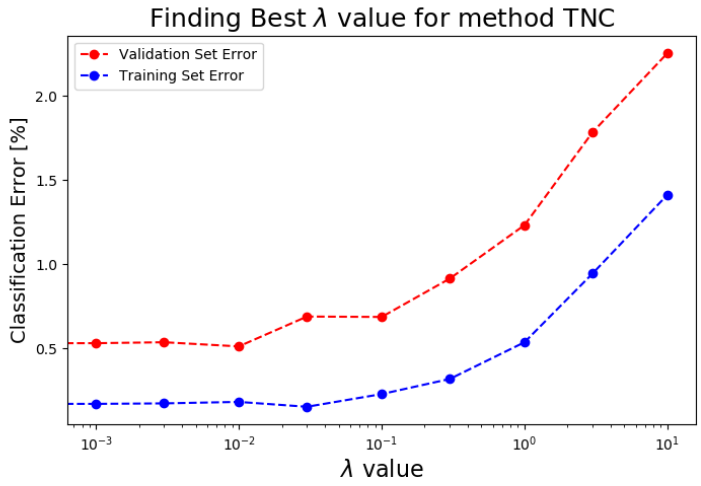


Ilustración 20. Error de clasificación para los distintos valores de lambda usando el algoritmo TNC

Redes Neuronales. Precisión de los algoritmos con cada una de las *lambdas*.

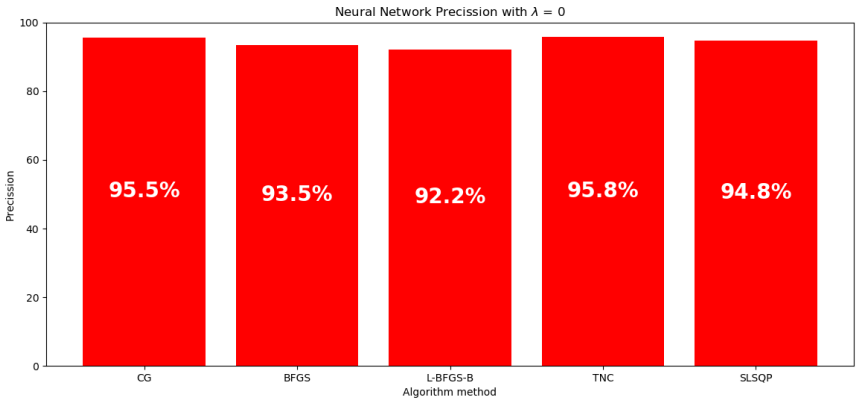


Ilustración 21. Precisión de la red neuronal para lambda 0

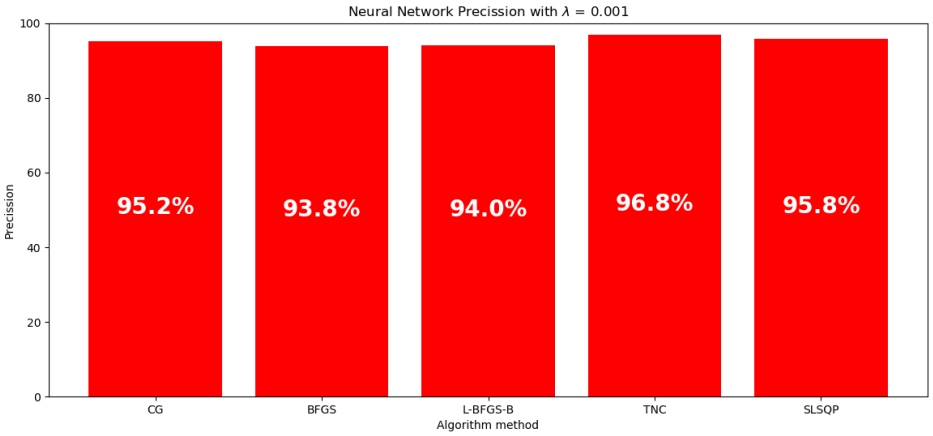


Ilustración 22. Precisión de la red neuronal para lambda 0.001

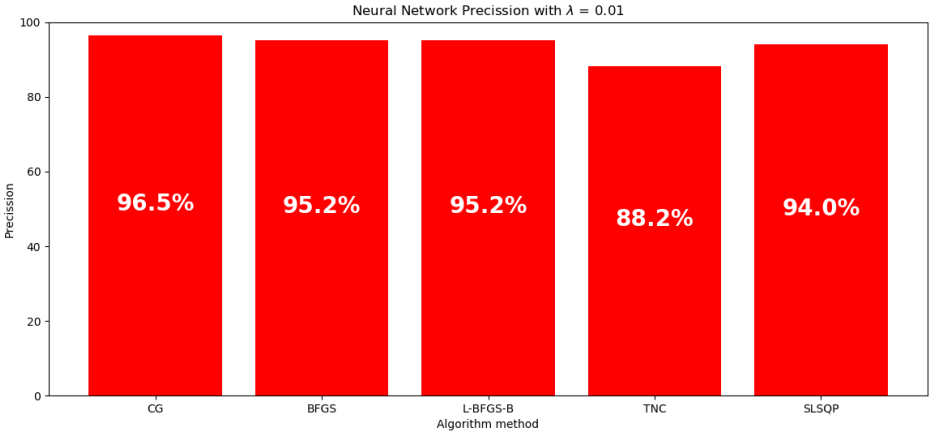


Ilustración 23. Precisión de la red neuronal para lambda 0.01

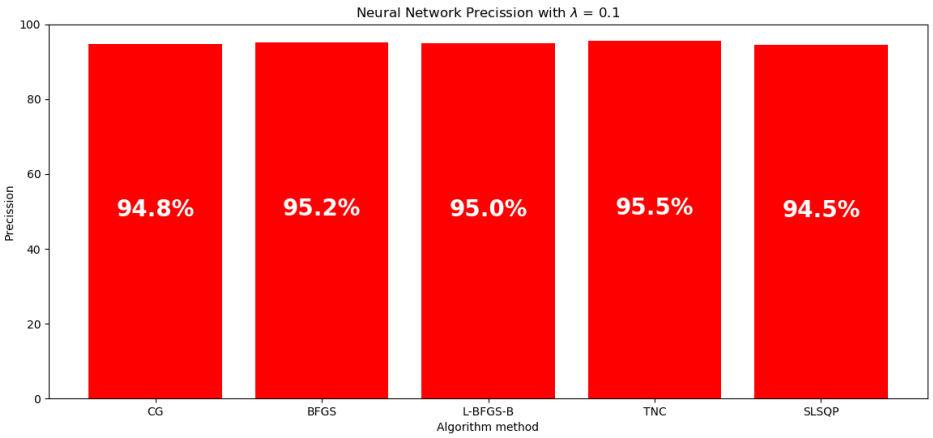


Ilustración 24. Precisión de la red neuronal para lambda 0.1

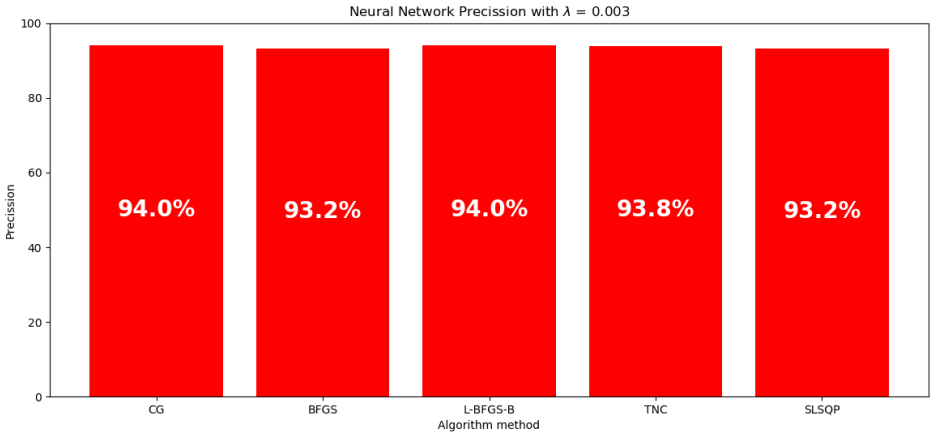


Ilustración 25. Precisión de la red neuronal para lambda 0.003

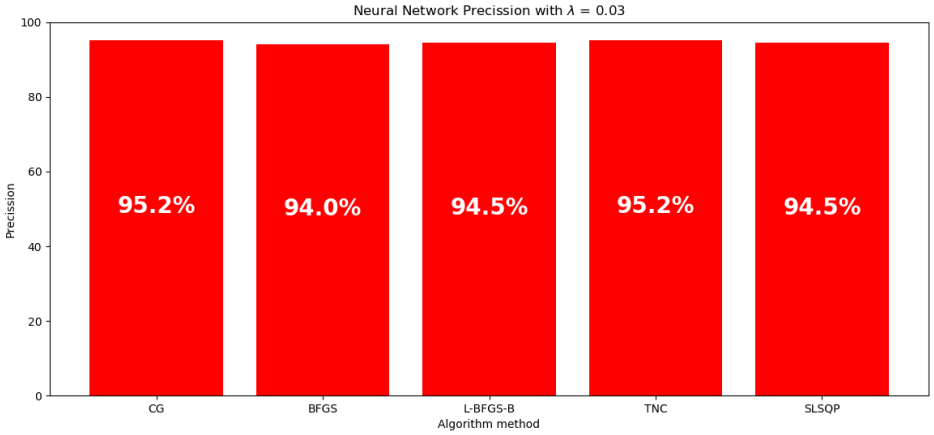


Ilustración 26. Precisión de la red neuronal para lambda 0.03

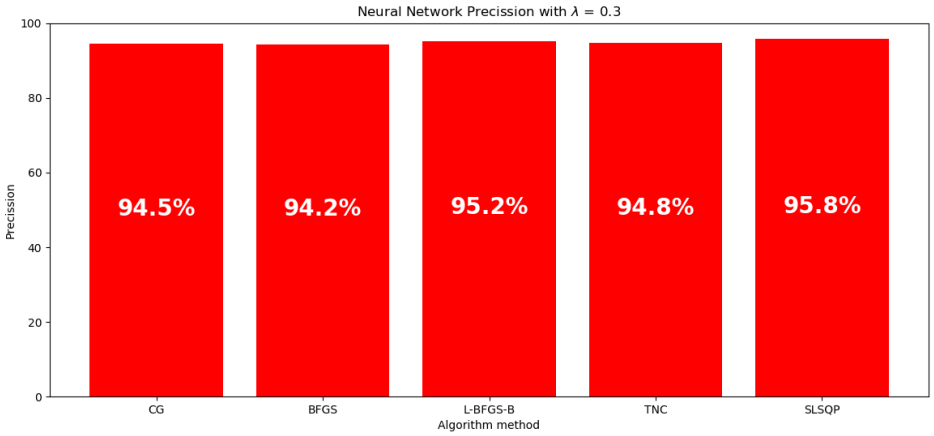


Ilustración 27. Precisión de la red neuronal para lambda 0.3

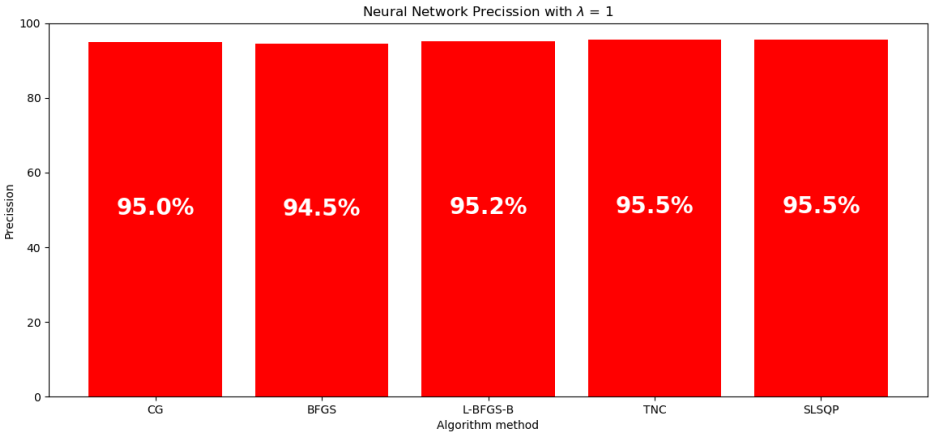


Ilustración 28. Precisión de la red neuronal para lambda 1

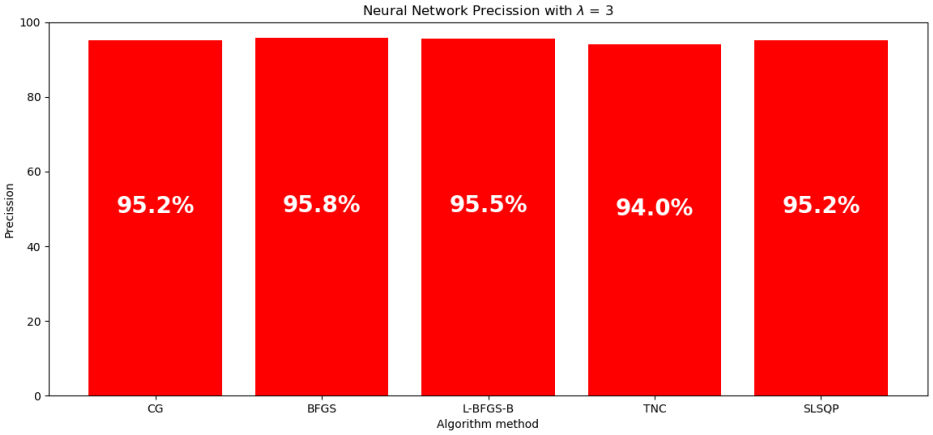


Ilustración 29. Precisión de la red neuronal para lambda 3

Redes Neuronales. Precisión de los algoritmos con el mejor valor de *lambda* para cada uno de ellos. Modificando el tamaño de la capa oculta.

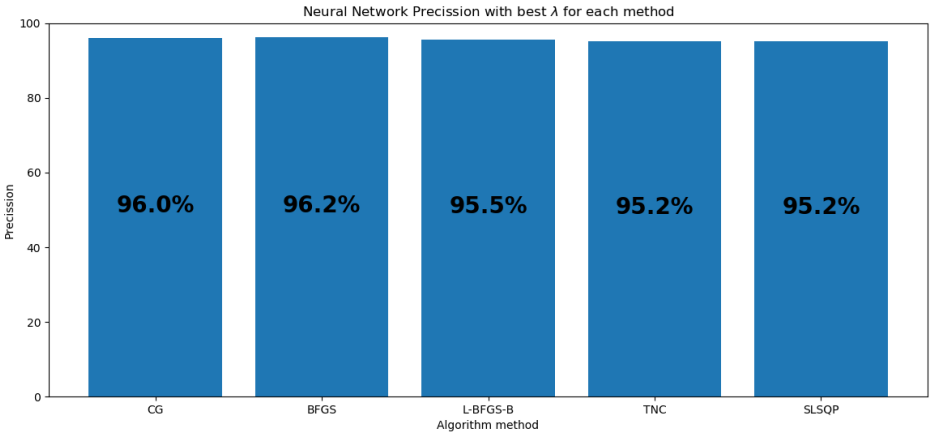


Ilustración 30. Precisión de los algoritmos con una capa oculta de tamaño 8

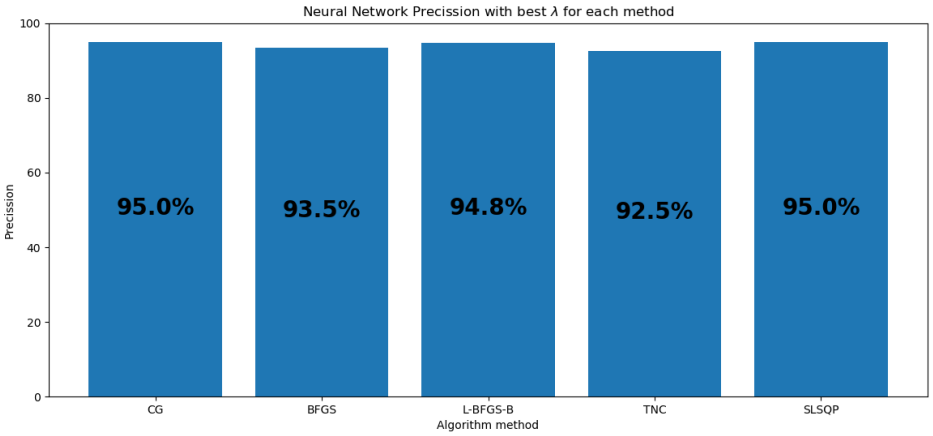


Ilustración 31. Precisión de los algoritmos con una capa oculta de tamaño 16

Support Vector Machines. Matrices de confusión.

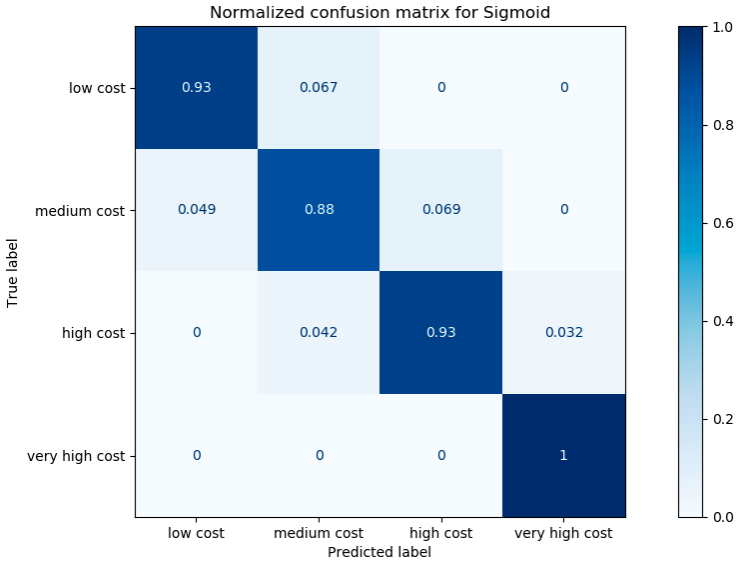


Ilustración 32. Matriz de confusión, Sigmoide

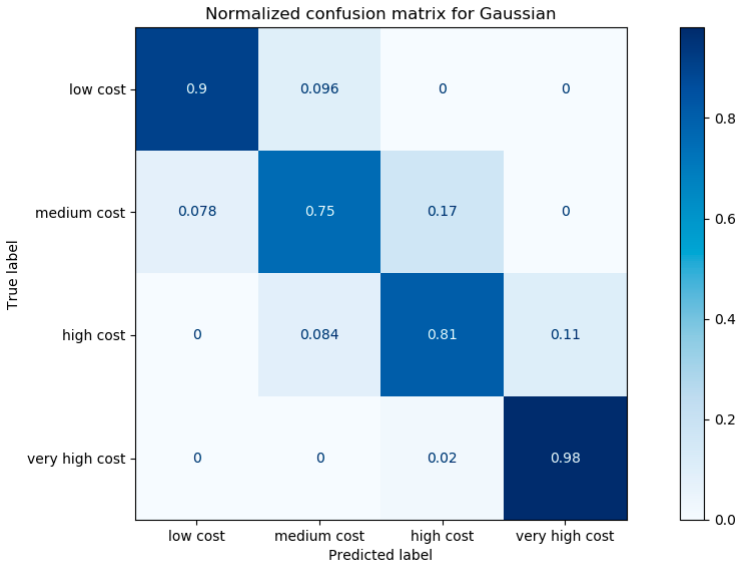


Ilustración 33. Matriz de confusión, Gaussiano

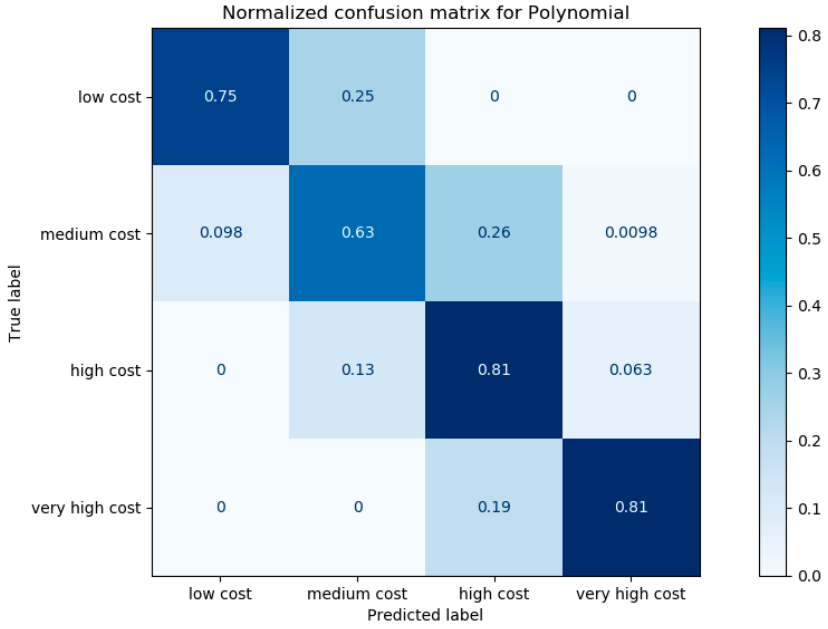


Ilustración 34. Matriz de confusión, Polinómico

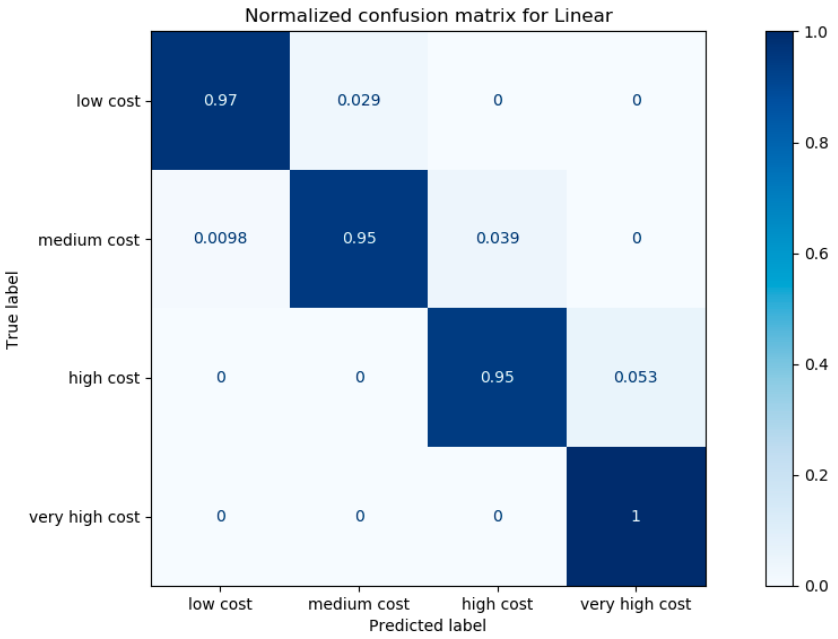


Ilustración 35. Matriz de confusión, Lineal

Support Vector Machines. Búsqueda del valor óptimo de C.

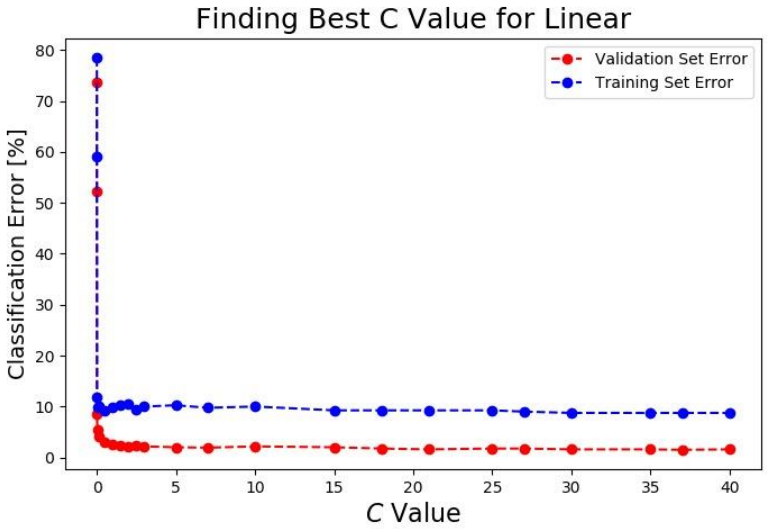


Ilustración 36. Mejor valor de C para el kernel Lineal

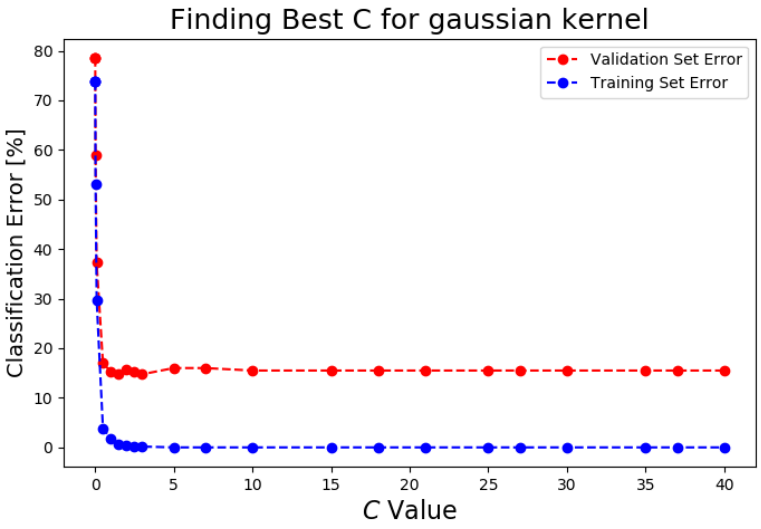


Ilustración 37. Mejor valor de C para el kernel Gaussiano

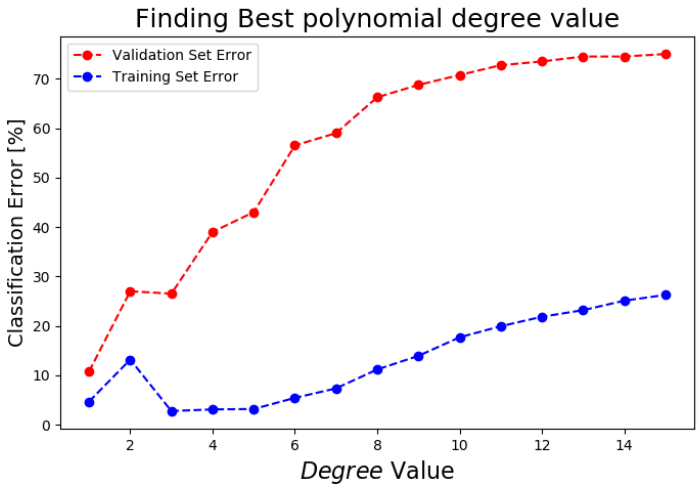


Ilustración 38. Mejor valor de C para el kernel Polinómico

Support Vector Machines. Fronteras de decisión.

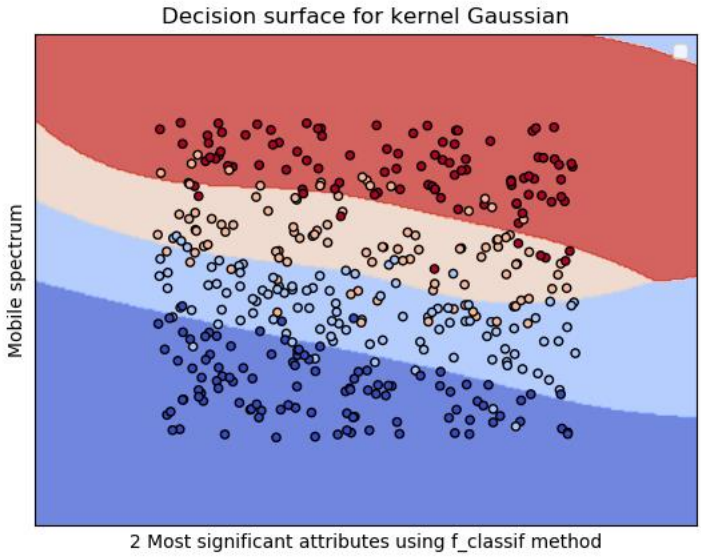


Ilustración 39. Frontera de decisión para el kernel Gaussiano

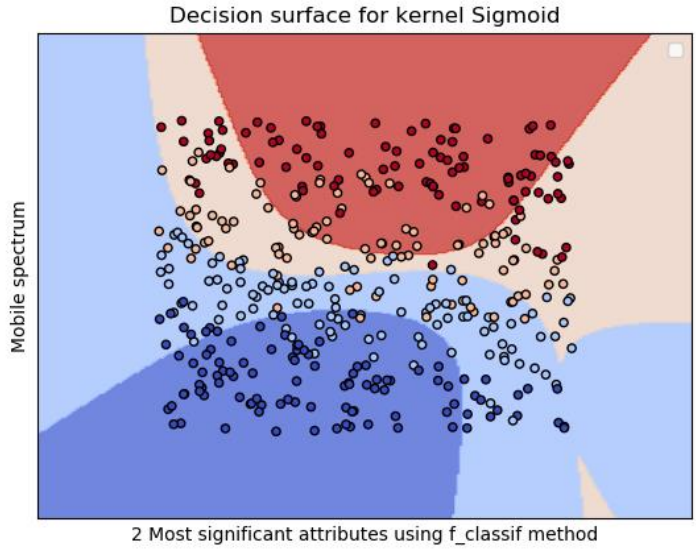


Ilustración 40. Frontera de decisión para el kernel Sigmoide

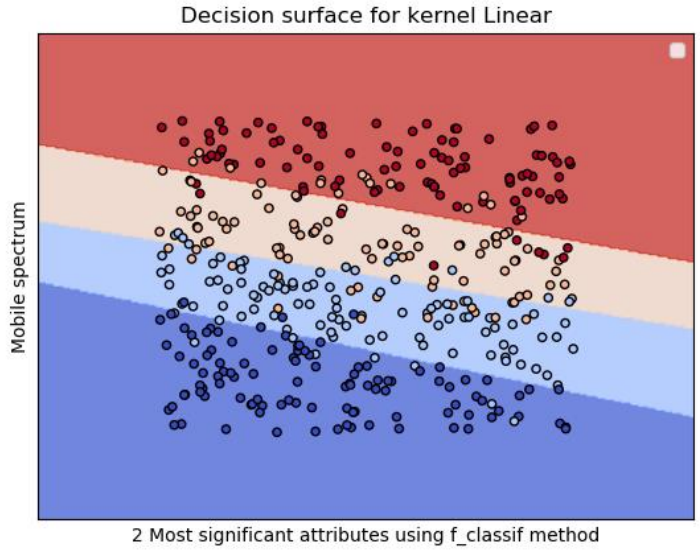


Ilustración 41. Frontera de decisión para el kernel Lineal

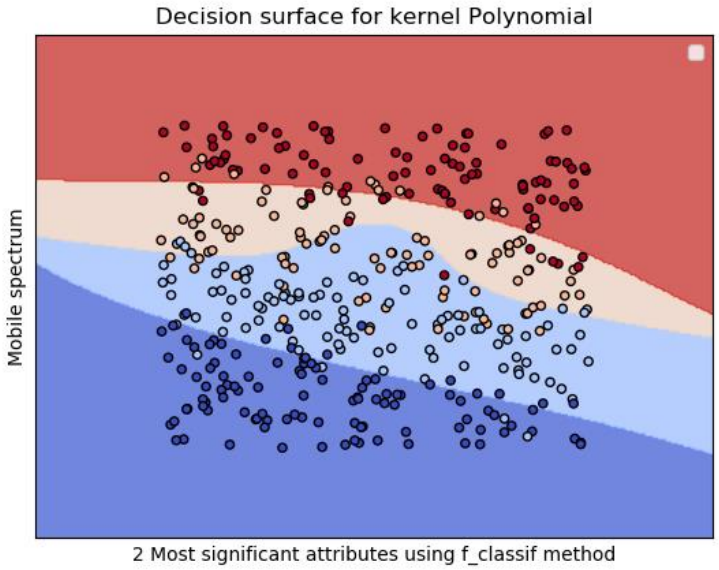


Ilustración 42. Frontera de decisión para el kernel Polinómico

Support Vector Machines. Porcentaje de efectividad de cada kernel.

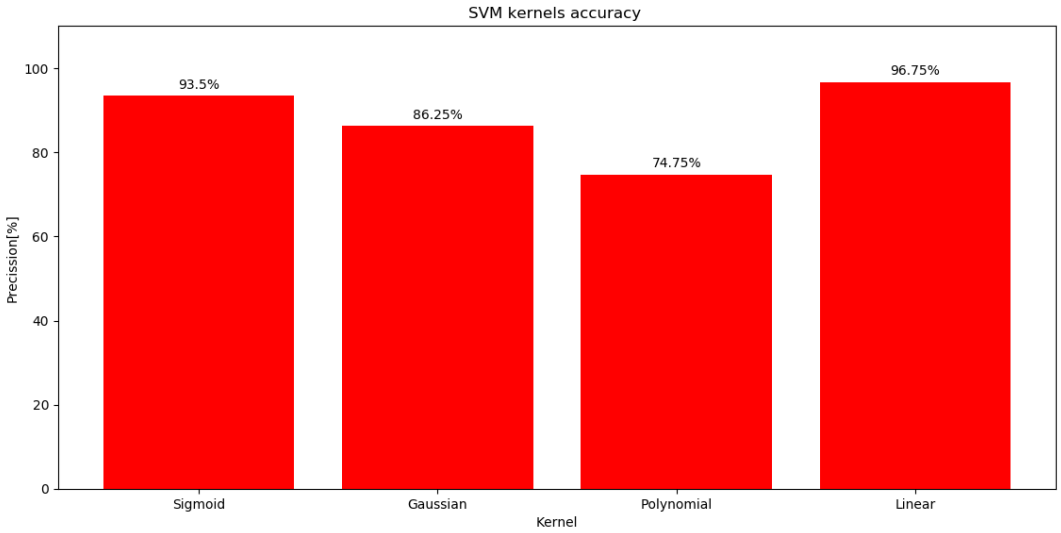


Ilustración 43. Precisión de cada kernel

Keras

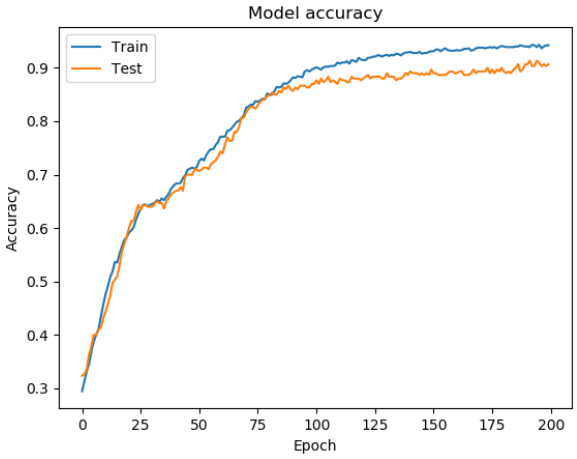


Ilustración 44. Keras, Regresión Logística. Precisión con lambda por defecto

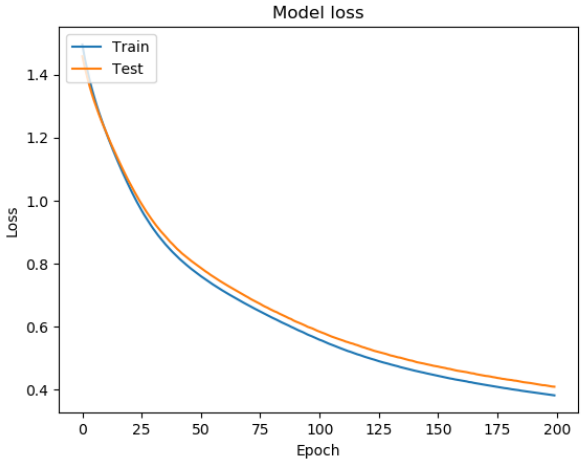


Ilustración 45. Keras, Regresión Logística. Pérdida con lambda por defecto

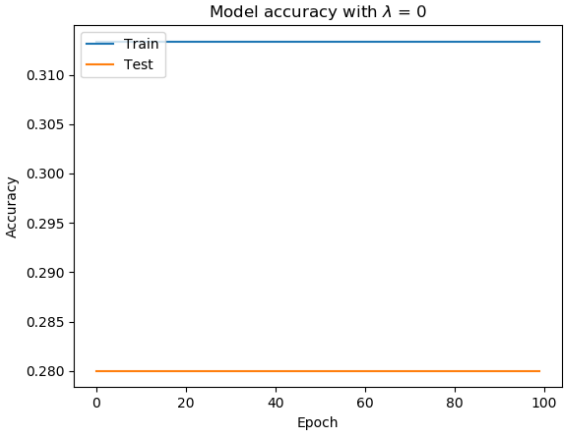


Ilustración 46. Keras, Regresión Logística. Precisión con lambda 0

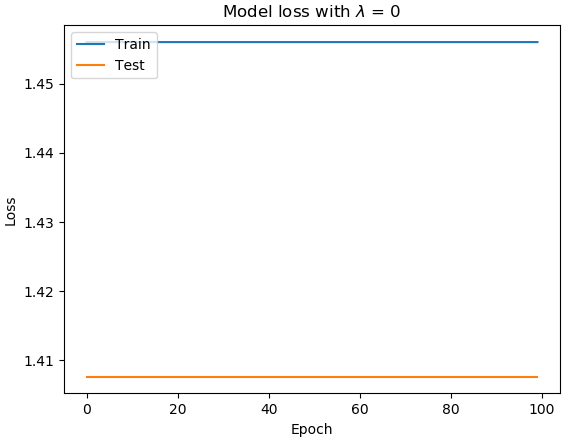


Ilustración 47. Keras, Regresión Logística. Pérdida con lambda 0

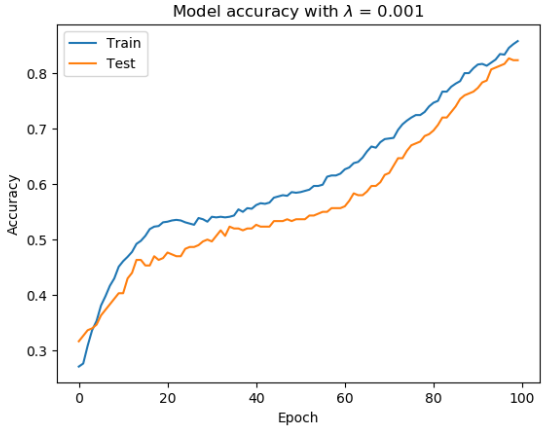


Ilustración 48. Keras, Regresión Logística. Precisión con lambda 0.001

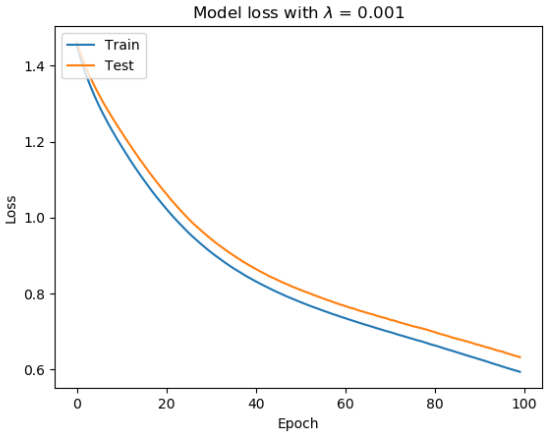


Ilustración 49. Keras, Regresión Logística. Pérdida con lambda 0.001

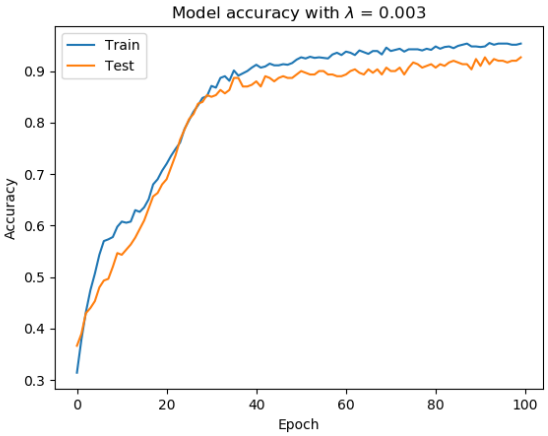


Ilustración 50. Keras, Regresión Logística. Precisión con lambda 0.003

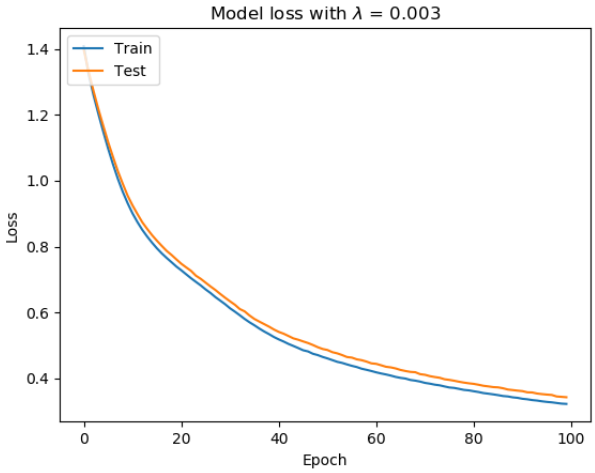


Ilustración 51. Keras, Regresión Logística. Pérdida con lambda 0.003

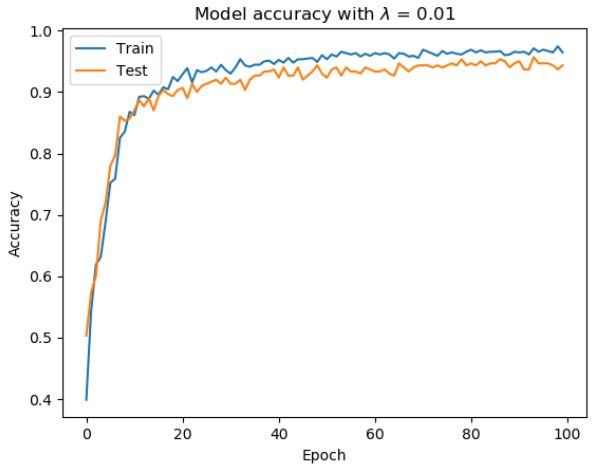


Ilustración 52. Keras, Regresión Logística. Precisión con lambda 0.01

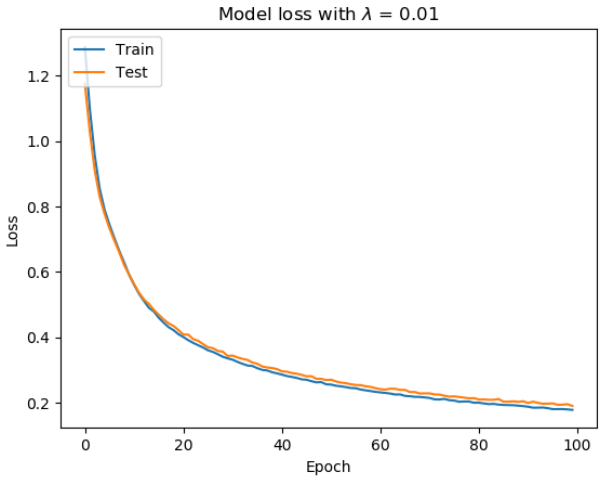


Ilustración 53. Keras, Regresión Logística. Pérdida con lambda 0.01

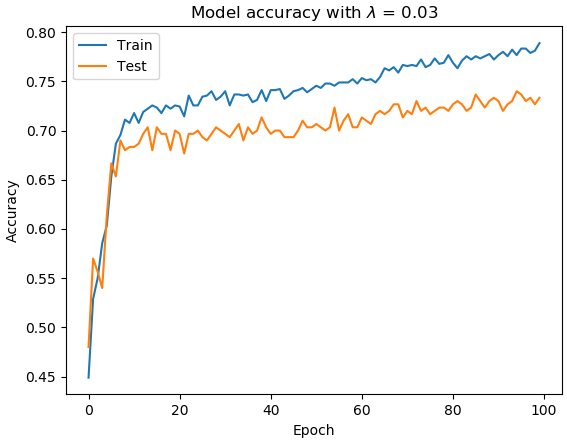


Ilustración 54. Keras, Regresión Logística. Precisión con lambda 0.03

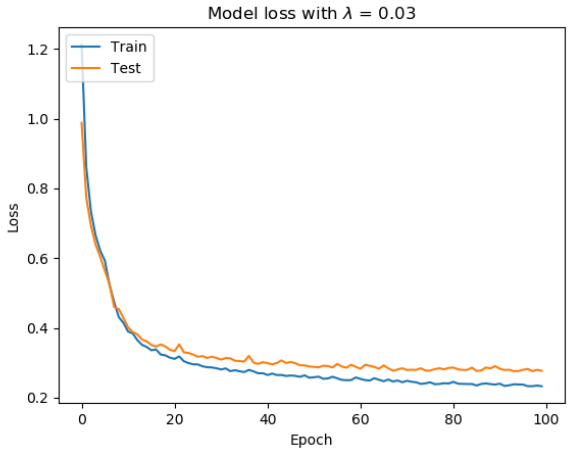


Ilustración 55. Keras, Regresión Logística. Pérdida con lambda 0.03

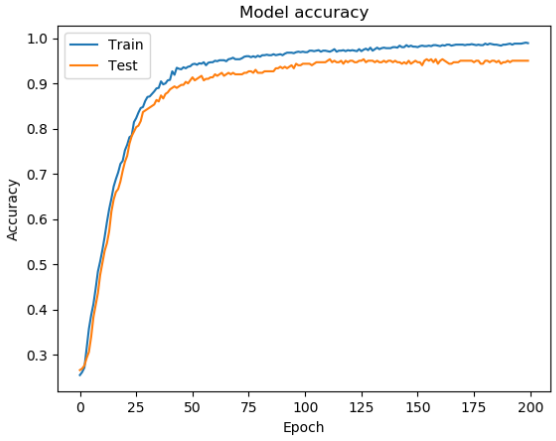


Ilustración 56. Keras, Redes Neuronales. Precisión con lambda por defecto

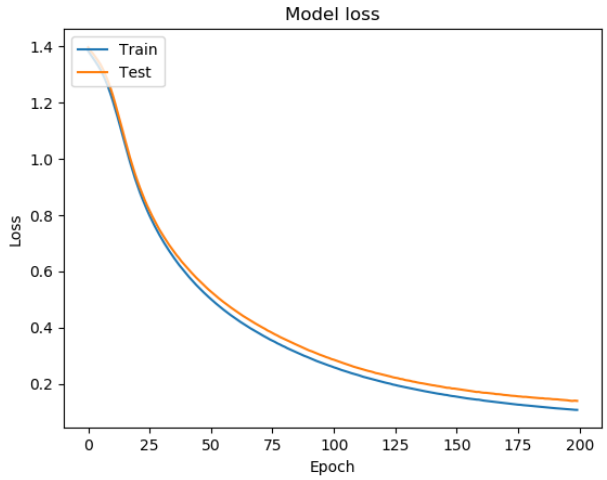


Ilustración 57. Keras, Redes Neuronales. Pérdida con lambda por defecto

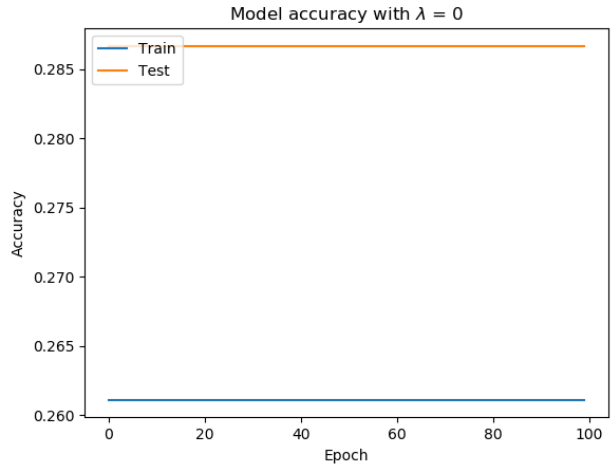


Ilustración 58. Keras, Redes Neuronales. Precisión con lambda 0

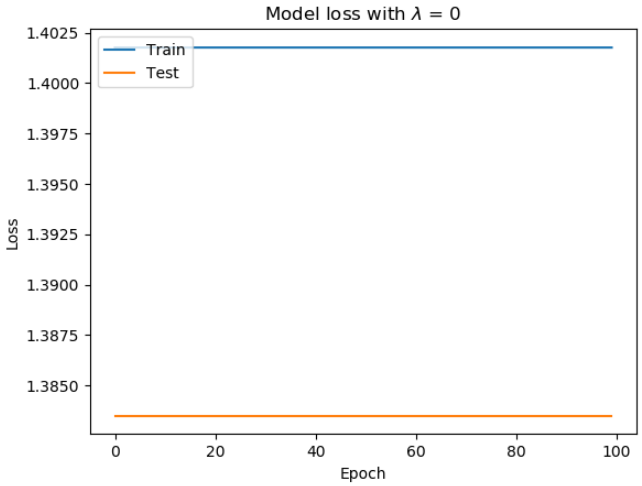


Ilustración 59. Keras, Redes Neuronales. Pérdida con lambda 0

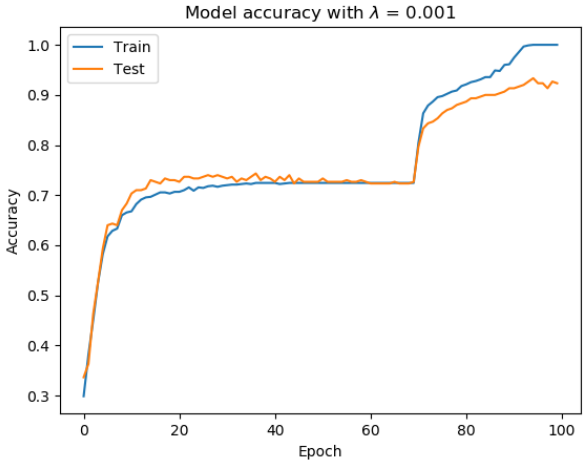


Ilustración 60. Keras, Redes Neuronales. Precisión con lambda 0.001

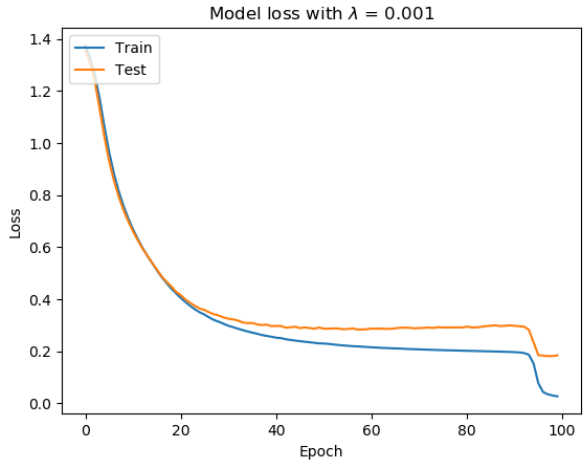


Ilustración 61. Keras, Redes Neuronales. Pérdida con lambda 0.001

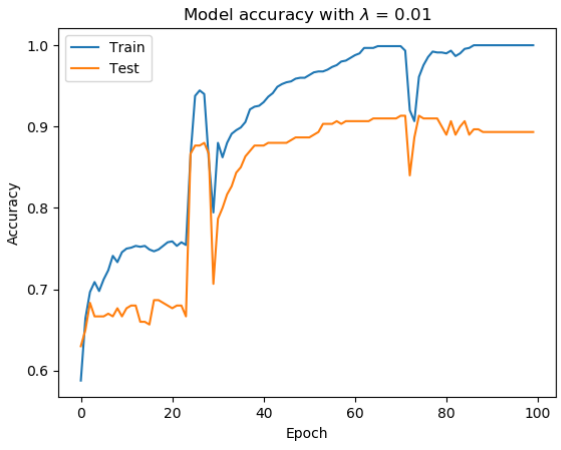


Ilustración 62. Keras, Redes Neuronales. Precisión con lambda 0.01

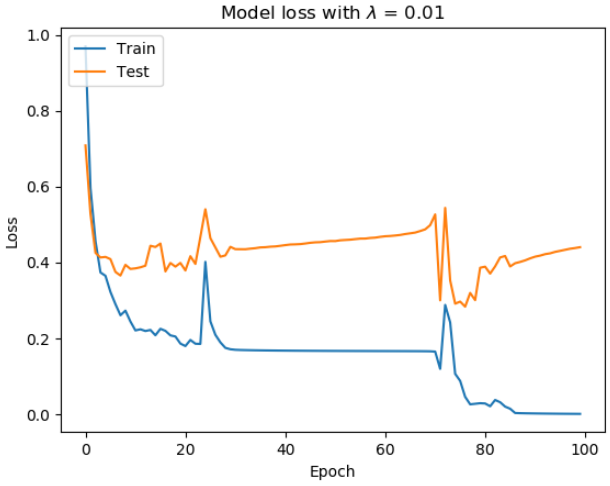


Ilustración 63. Keras, Redes Neuronales. Pérdida con lambda 0.01

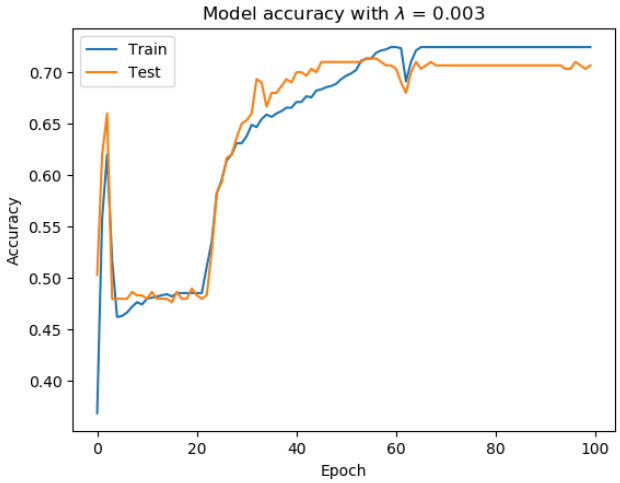


Ilustración 64. Keras, Redes Neuronales. Precisión con lambda 0.003

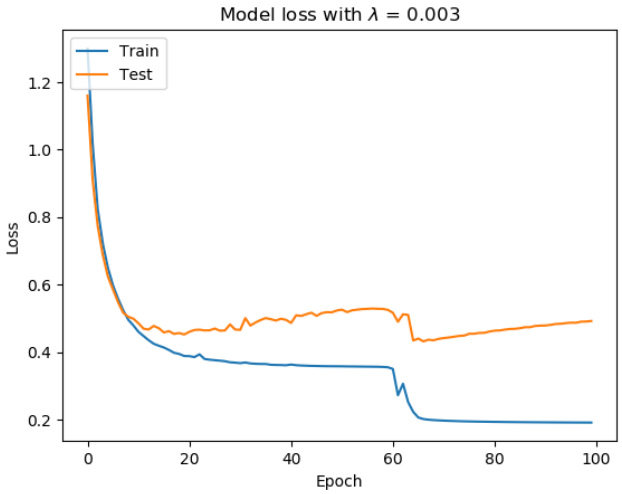


Ilustración 65. Keras, Redes Neuronales. Pérdida con lambda 0.003

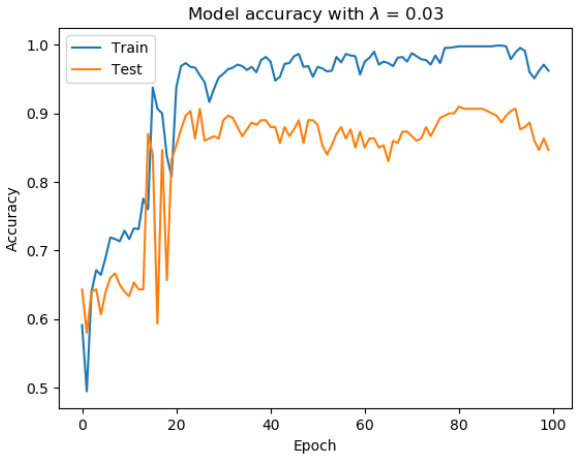


Ilustración 66. Keras, Redes Neuronales. Precisión con lambda 0.03

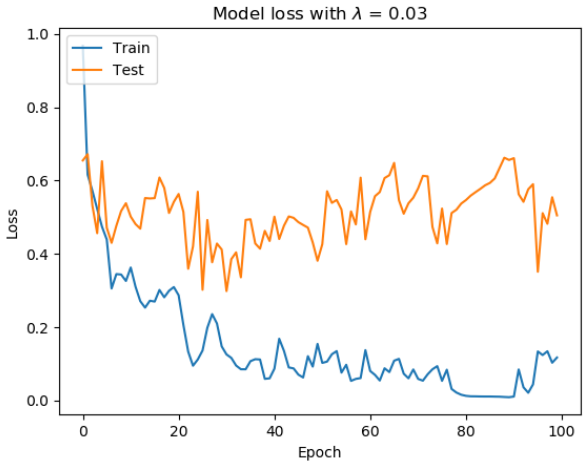
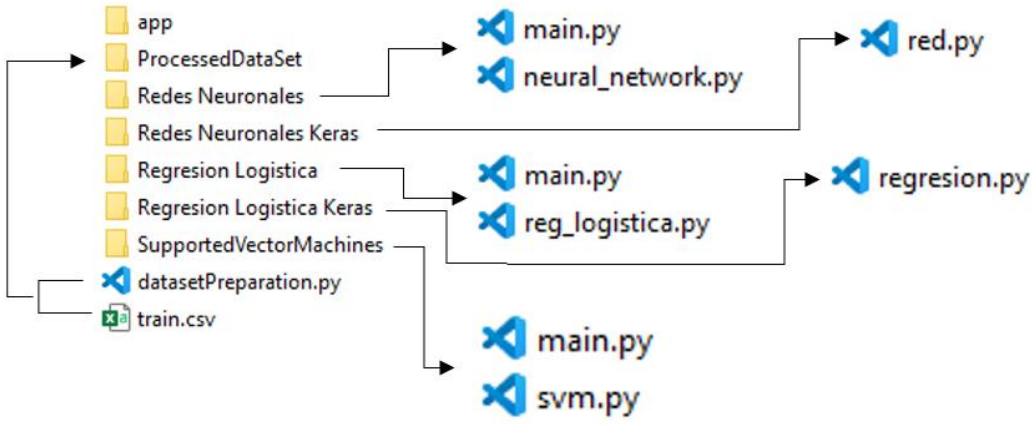


Ilustración 67. Keras, Redes Neuronales. Pérdida con lambda 0.03

Apéndice C. Imágenes

Estructura de directorios y ficheros principal



Estructura de directorios y ficheros de Phone CompluHouse