EJERCICIO 1:

```
clearvars
close all
clc
load CENSUSUSA1994.mat
Data = adultdata;
clearvars -except Data
```

POR REGRESIÓN LOGÍSTICA:

Las variables categóricas ofrecen ventajas al trabajar con problemas de clasificación. Viendo que Data.salary es de hecho una columna categorizada, estamos listos para resolver este ejercicio.

```
RLmodel = fitglm(Data,'linear','Distribution','binomial', ...
"PredictorVars",[ "age" "education" "occupation" "race" "sex" "hours_per_week"],...
"ResponseVar","salary")
```

```
Warning: Iteration limit reached.
RLmodel =
Generalized linear regression model:
    logit(P(salary='>50K')) ~ 1 + age + education + occupation + race + sex + hours_per_week
    Distribution = Binomial

Estimated Coefficients:
```

	Estimate	SE	tStat	pVa
(Intercept)	-7.13061242086773	0.264523651914936	-26.9564266531476	4.79589987
age	0.0448713881743225	0.0013121637591053	34.1964848998102	2.72706659
education_11th	0.0945928098144779	0.199772866313529	0.47350179010808	0.6358
education_12th	0.398604284703484	0.245389829771734	1.62437165824791	0.1042
education_1st-4th	-0.737021273966263	0.45372350205003	-1.62438416929303	0.1042
education_5th-6th	-0.358503161585267	0.316079401449324	-1.13421868031076	0.2567
education_7th-8th	-0.498407466788762	0.22595068035314	-2.2058241471537	0.02739
education_9th	-0.257106610525344	0.253429722110928	-1.01450851298652	0.3103
education_Assoc-acdm	1.38490454783043	0.163773906410291	8.45619780455701	2.762368
education_Assoc-voc	1.41493358405233	0.158392574713787	8.93308026976069	4.143088
education_Bachelors	1.87055071622928	0.148004643002753	12.6384597015277	1.295672
education_Doctorate	2.71413979510957	0.192611084122804	14.091295978476	4.296046
education_HS-grad	0.869157760368103	0.145062122308713	5.9916244608528	2.077552
education_Masters	2.17789108294134	0.155636024867421	13.9934895201582	1.708204
education_Preschool	-99.4255516827659	9894662.18349294	-1.00484028498351e-05	0.9999
education_Prof-school	2.82311523846971	0.181137369238251	15.5854932107159	9.135552
education_Some-college	1.1484837357531	0.146826379248954	7.82205310536039	5.196867
occupation_Armed-Forces	-0.690304857745221	1.1174533120263	-0.617748276653706	0.5367
occupation_Craft-repair	0.0912169692851779	0.0681677531430334	1.33812492093998	0.1808
occupation_Exec-managerial	0.827523130357884	0.0640950232067721	12.9108796433114	3.908108
occupation_Farming-fishing	-1.15453245575329	0.121623671364813	-9.4926624299166	2.252073
occupation_Handlers-cleaners	-0.868276853039717	0.127966236919392	-6.78520267487948	1.159236
occupation_Machine-op-inspct	-0.224610784000619	0.0899399718160725	-2.49734105387479	0.01251
occupation_Other-service	-1.10438863216205	0.105096370256023	-10.5083422907154	7.907117
occupation_Priv-house-serv	-2.96383688276122	1.10601240075184	-2.67975013729184	0.007367
occupation_Prof-specialty	0.48040345113444	0.0677961760230829	7.08599628054058	1.380475
occupation_Protective-serv	0.422108059490366	0.105763532736774	3.99105484251282	6.578006
occupation_Sales	0.259436619130997	0.0681151579518002	3.80879420869258	0.000139
occupation_Tech-support	0.551534150350612	0.094414540247716	5.84162300535011	5.169468
occupation_Transport-moving	-0.115348242982322	0.0875599133512422	-1.31736360358888	0.1877
race_Asian-Pac-Islander	0.458631906372229	0.215565839028092	2.12757229271593	0.03337
race_Black	0.232177073635175	0.207220681845482	1.12043388510951	0.2625
race_Other	-0.160735010755579	0.317974002177248	-0.505497335175158	0.6132

```
race_White          0.560140078750909    0.19841767715684     2.823035158849     0.004757
sex_Male            1.17871139188939     0.0414564331726762   28.4325327019759   8.01351966
hours_per_week      0.0321247155269353   0.00143399138724199  22.4023071635884   3.73715320

30718 observations, 30682 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 8.68e+03, p-value = 0
```

En este caso usaremos la distribución binominal, por haber dos posibilidades. Es decir, estamos en el caso donde usaremos un modelo de salida con dos posibilidades: "<=50K" y ">50K"

```
RLscores = predict(RLmodel,Data)
```

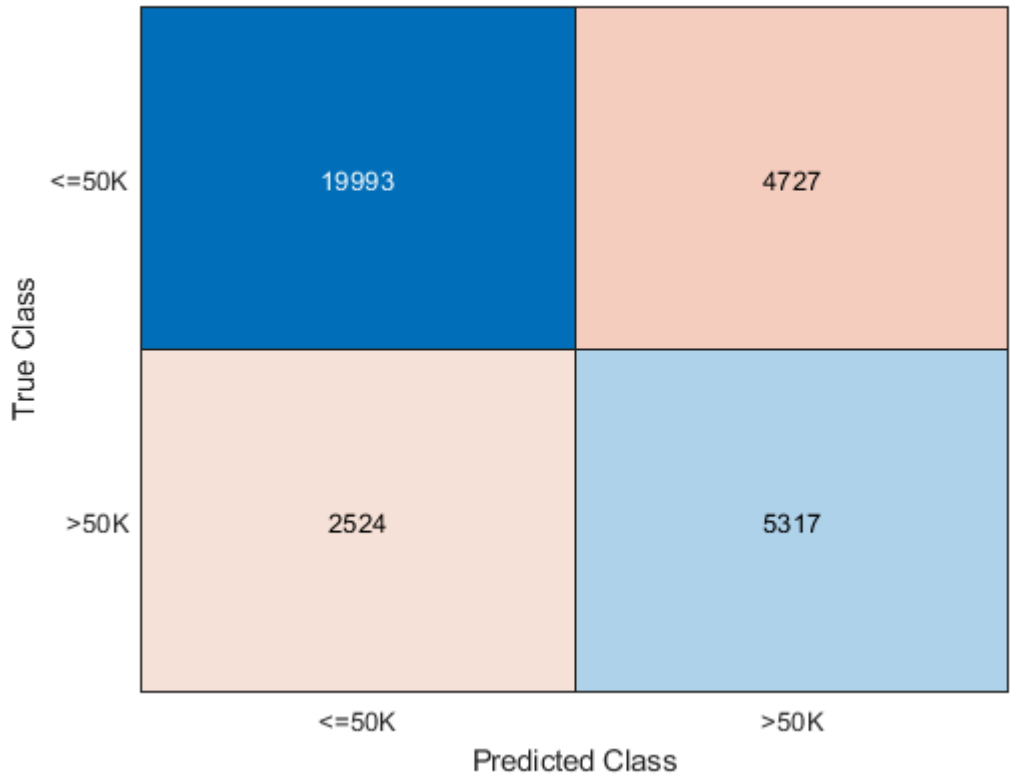
```
RLscores = 32561×1
   0.380780322283496
   0.491876616587200
   0.083113338191513
   0.055711991399864
   0.118564606829736
   0.349820538105809
   0.003883075614237
   0.520948810220870
   0.285992271600663
   0.616779210846604
     ⋮
```

Ahora bien, lo que es de interés es convertir este valor numérico a una determinada clase: "<=50K" o ">50K".

```
RLumb = 0.3;
predictRL = RLscores >= RLumb;
predictRL = categorical(predictRL, [false true], ["<=50K" ">50K"]);
Data.predictLinearRL = predictRL;
```

Como es natural, nos encontraremos con datos mal clasificados. Usaremos la teoría de la matriz de precisión y métricas de rendimiento para conocer la calidad de los resultados predichos anteriormente (recuerde que lo último mencionado depende de la frontera, o umbral, de decisión previamente tomado). Escogeremos la métrica accuracy como parámetro determinante de esta calidad.

```
confusionchart(Data.salary,predictRL)
```



Sujeto a:

```
metrics = confusionmat(Data.salary,predictRL)
```

```
metrics = 2×2
       19993        4727
        2524        5317
```

```
accuracy = trace(metrics)/sum(metrics,'all')
```

```
accuracy =
   0.777310279168330
```

POR KNN:

```
NumNeighbors = 30;
Data.age = categorical(Data.age);
Data.hours_per_week = categorical(Data.hours_per_week);
KNNmodel = fitcknn(Data,"salary","NumNeighbors",NumNeighbors,"DistanceWeight","equal",...
"PredictorNames",[ "age" "education" "occupation" "race" "sex" "hours_per_week"],...
"ResponseName","salary")
```

```
KNNmodel =
  ClassificationKNN
             PredictorNames: {'age'  'education'  'occupation'  'race'  'sex'  'hours_per_week'}
               ResponseName: 'salary'
      CategoricalPredictors: [1 2 3 4 5 6]
                 ClassNames: [<=50K    >50K]
             ScoreTransform: 'none'
            NumObservations: 32561
```

```
                Distance: 'hamming'
            NumNeighbors: 30

  Properties, Methods
```

```
KNNumb = 0.3;
[KNNpredict,KNNscores] = predict(KNNmodel,Data)
```

```
KNNpredict = 32561×1 categorical
<=50K
>50K
<=50K
<=50K
<=50K
<=50K
<=50K
>50K
<=50K
>50K
    ⋮

KNNscores = 32561×2
   0.633333333333333   0.366666666666667
   0.200000000000000   0.800000000000000
   0.833333333333333   0.166666666666667
   0.933333333333333   0.066666666666667
   0.866666666666667   0.133333333333333
   0.566666666666667   0.433333333333333
   0.966666666666667   0.033333333333333
   0.466666666666667   0.533333333333333
   0.733333333333333   0.266666666666667
   0.266666666666667   0.733333333333333
      ⋮
```

Convertimos los scores a predicciones, basados en el umbral:

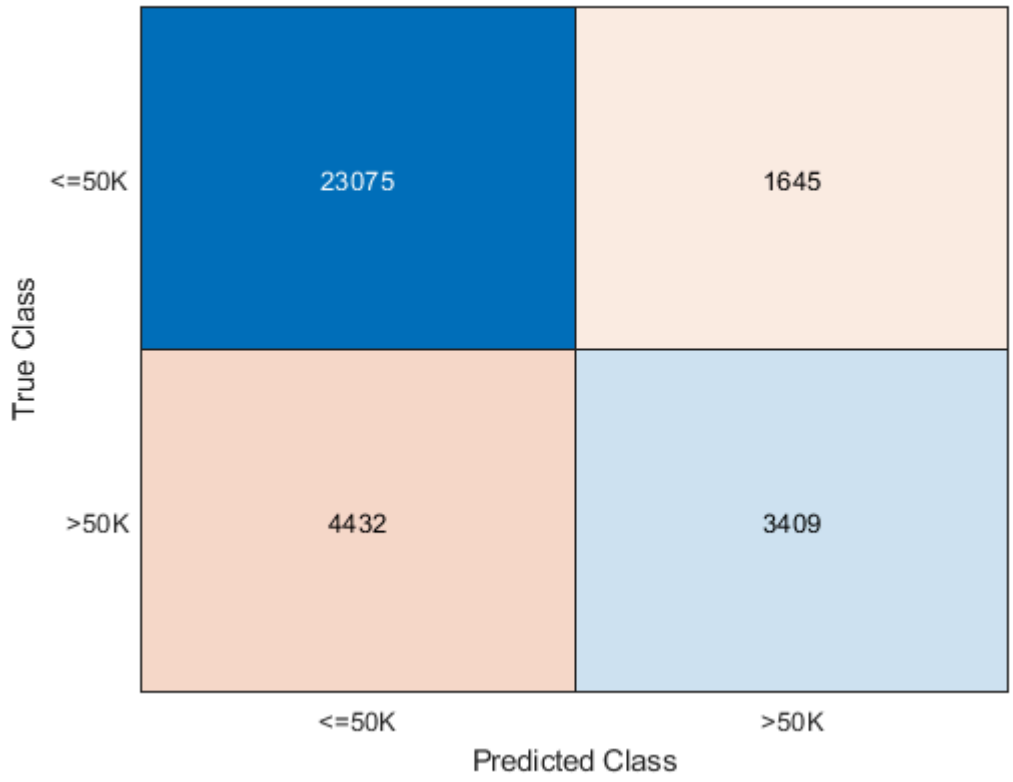
```
predictKNN = KNNscores(:,1) >= KNNumb;
predictKNN = categorical(predictKNN,[true false],["<=50K" ">50K"])
```

```
predictKNN = 32561×1 categorical
<=50K
>50K
<=50K
<=50K
<=50K
<=50K
<=50K
>50K
<=50K
>50K
    ⋮
```

```
Data.predictMediumKNN = predictKNN;
```

Se tiene, por tanto, la matriz de confusión:

```
confusionchart(Data.salary,predictKNN)
```



Sujeto a:

```
metrics = confusionmat(Data.salary,predictKNN)
```

```
metrics = 2×2
       23075        1645
        4432        3409
```

```
accuracy = trace(metrics)/sum(metrics,'all')
```

```
accuracy =
   0.813365682872148
```

Observamos que la precisión por KNN es mayor que por regresión logística.

EJERCICIO 2:

```
clearvars
close all
clc
```

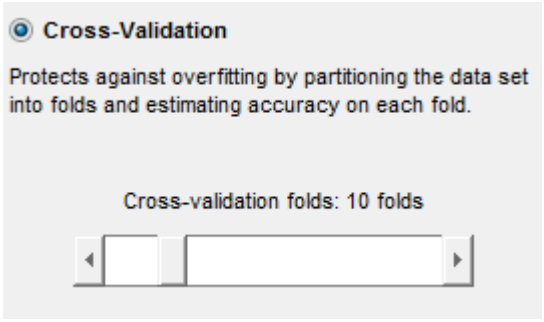
Importamos:

```
Data = importfile("C:\Users\pc\Desktop\MATLAB_CODES\PROCESAMIENTO DE SEÑALES Y DATOS\PCs\2022-1
```

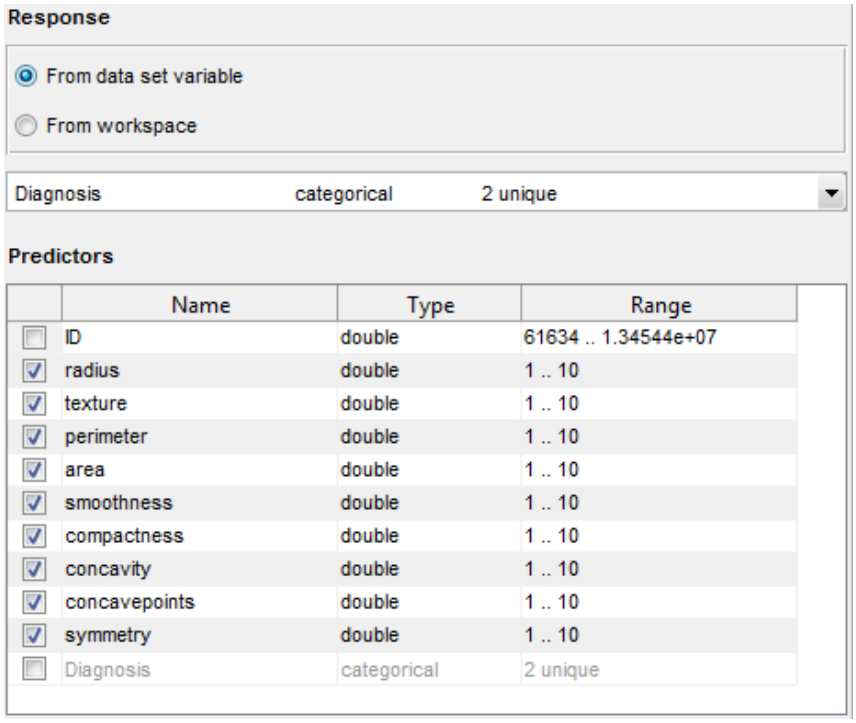
Hacemos los cambios que sugiere el problema:

```
Data.Diagnosis = categorical(Data.Diagnosis);
Data.Diagnosis(Data.Diagnosis == '2') = "Malignant";
Data.Diagnosis(Data.Diagnosis == '4') = "Benign";
```

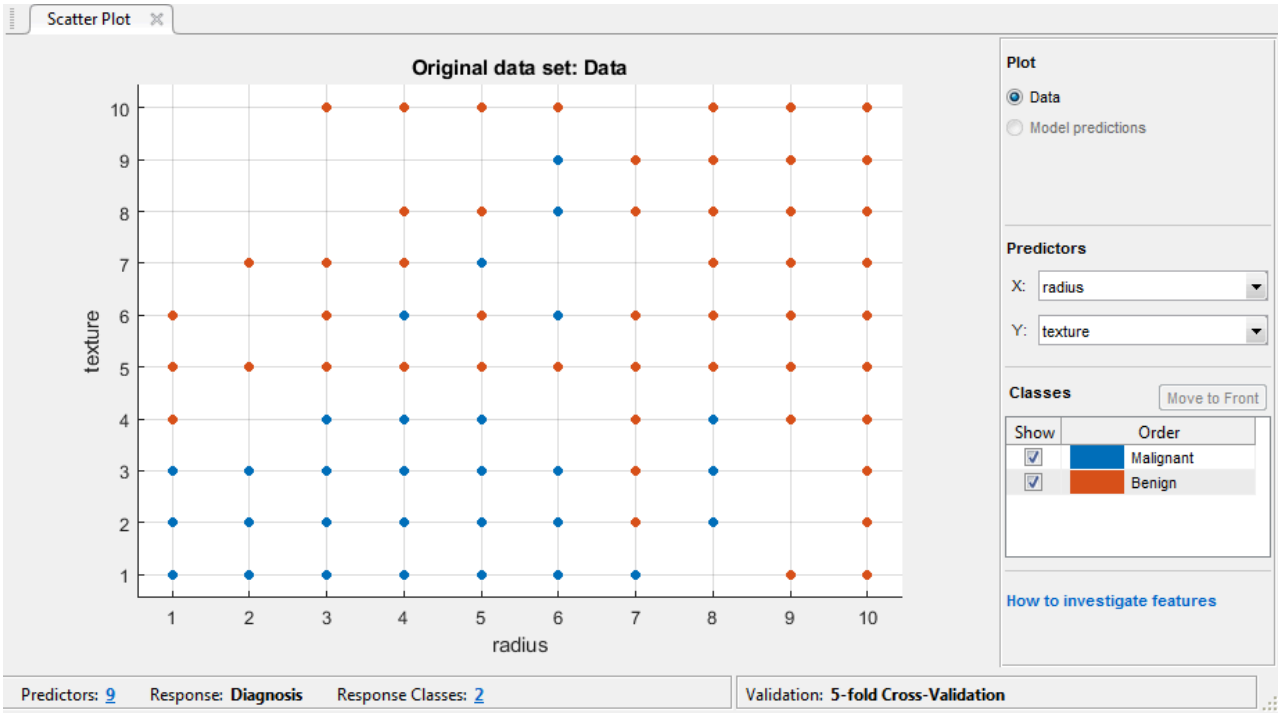
Resolveremos este problema haciendo uso de la aplicación Clasification Learner. Usamos el método de validación cruzada k-fold, con k = 10.



Usamos como variables predictoras del 2-10 y como variable de respuesta la variable 11 del listado del problema.

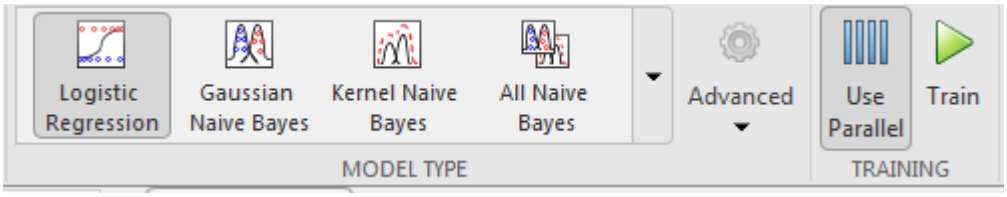


Obtenemos lo siguiente:

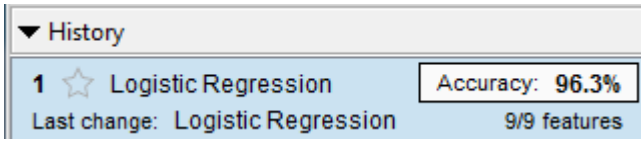


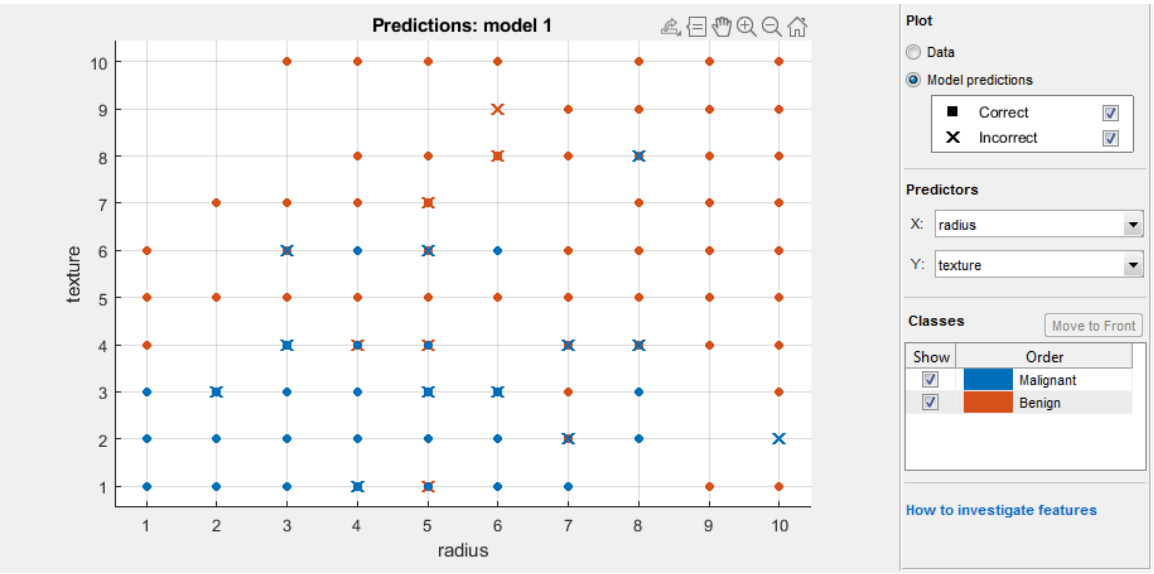
EVALUAMOS POR REGRESIÓN LOGÍSTICA:

Escogemos el modelo.

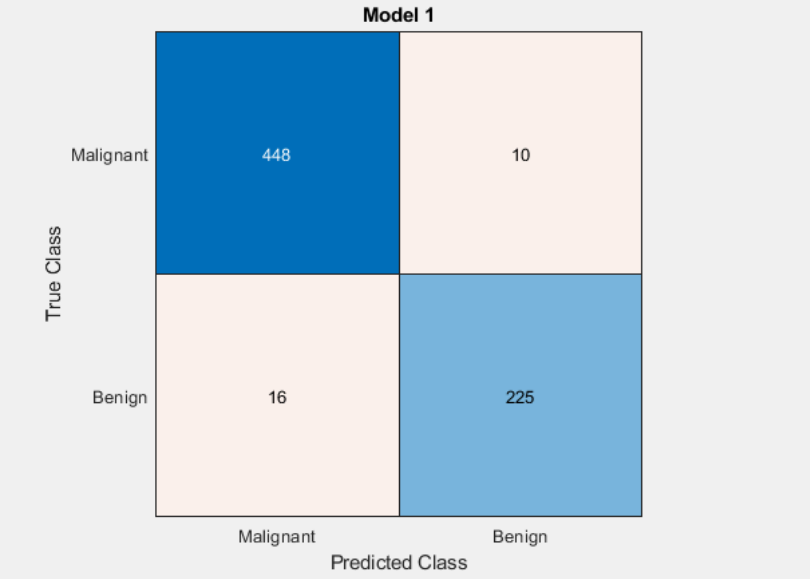


Al entrenar, tenemos (ejemplificando con el par radius-texture):

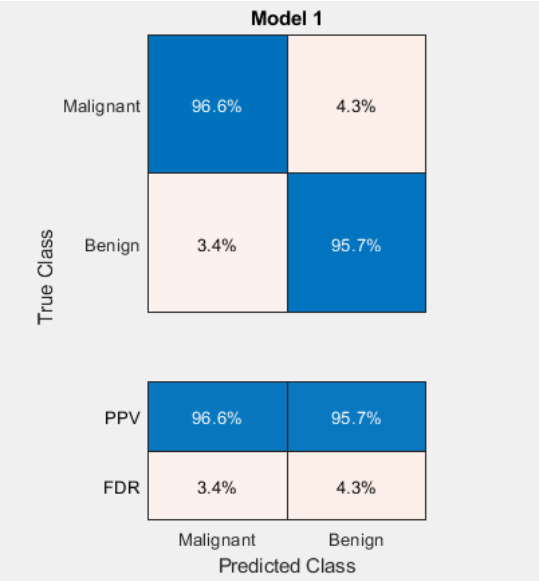




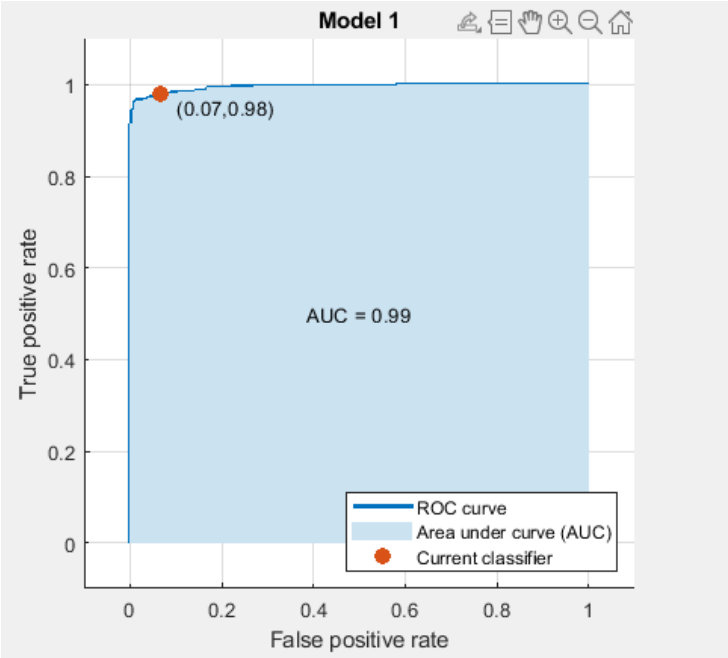
Donde las X representan las malas clasificaciones. Podemos ver, además, la matriz de confusión:



Donde la diagonal nos da el número de data correctamente clasificada.



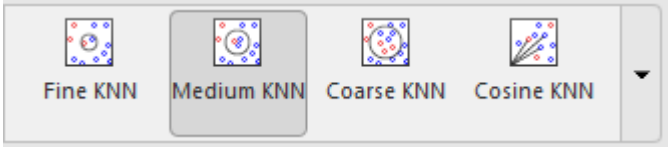
Sujeto también a la curva ROC (donde la clase positiva es Malignant):



Donde True positive rate representa a la métrica Recall, y False positive rate a Fallout.

EVALUAMOS POR KNN

Escogemos ahora:



Sujeto a:

Model 2: Draft

Model Type
Preset: Medium KNN
Number of neighbors: 30
Distance metric: Euclidean
Distance weight: Equal
Standardize data: true

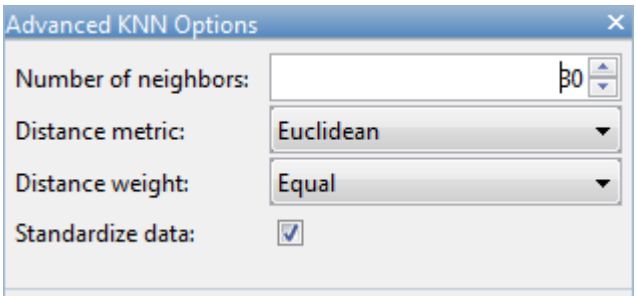
Optimizer Options
Hyperparameter options disabled

Feature Selection
All features used in the model, before PCA

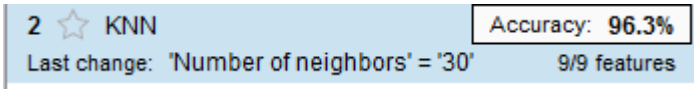
PCA
PCA disabled

Misclassification Costs
Cost matrix: default

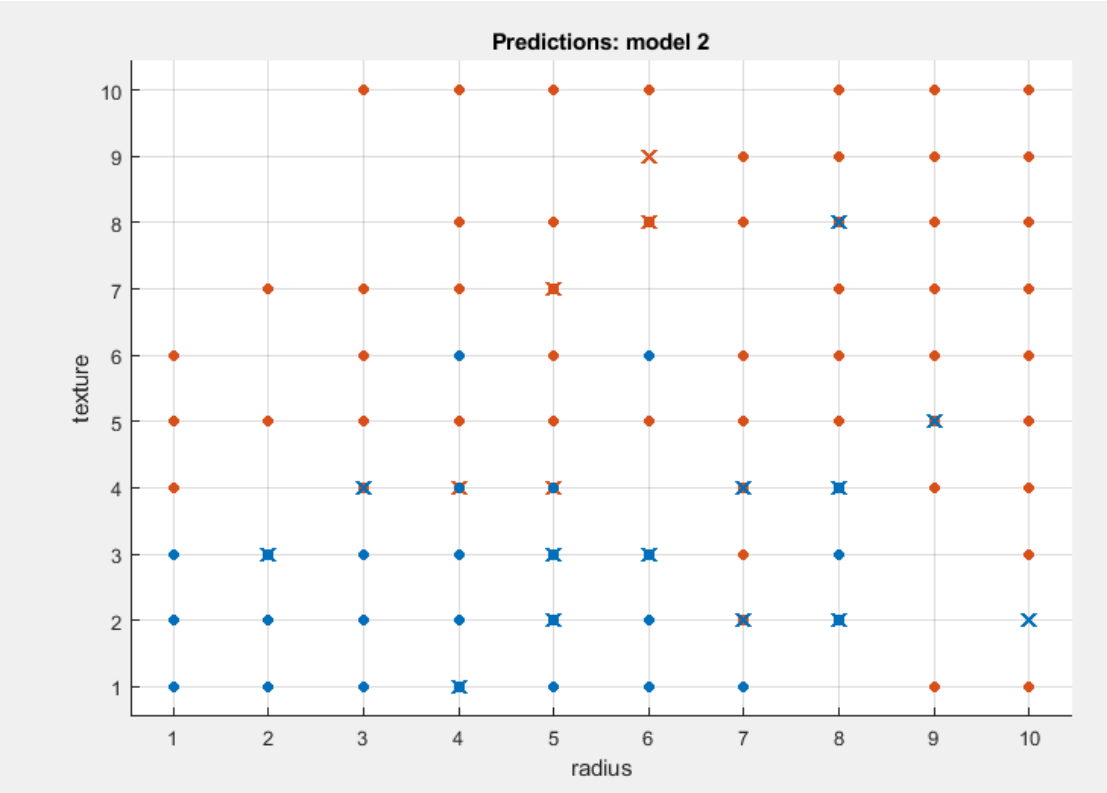
Donde se modificó el número de vecinos:



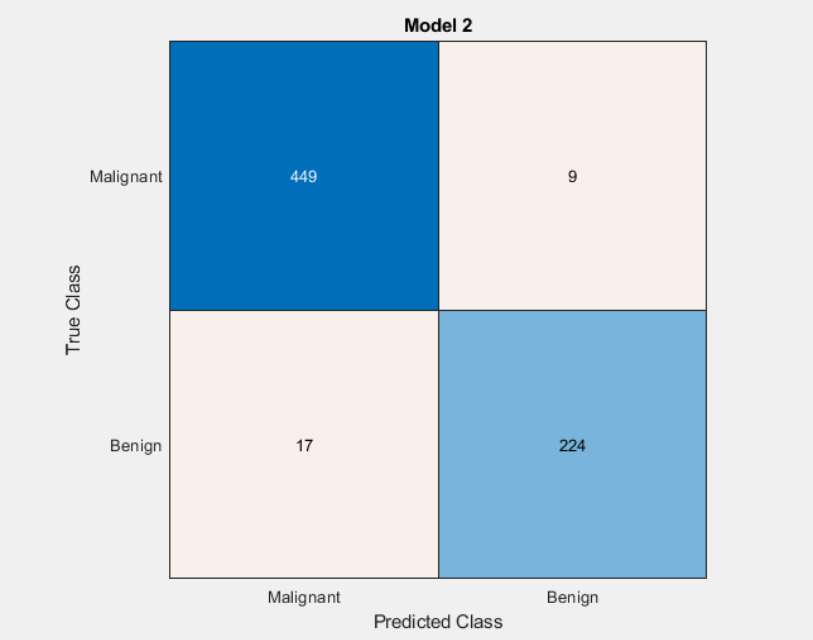
Una vez entrenado tenemos:

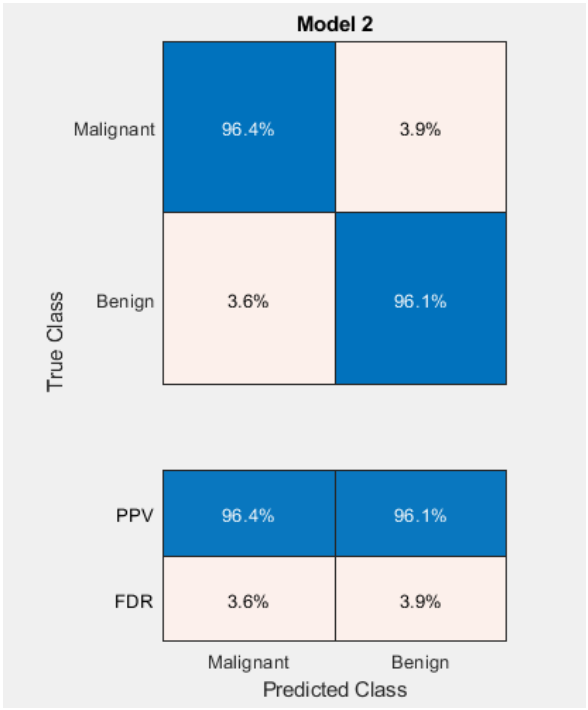


Y, por ejemplo:

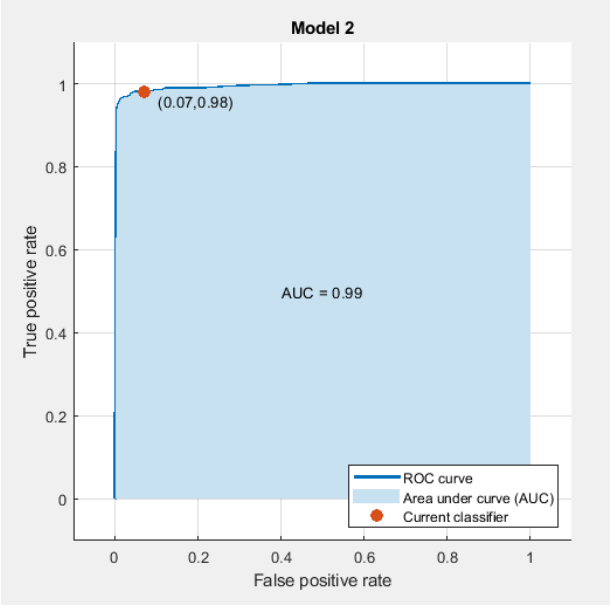

Predictions: model 2
texture
radius
1
2
3
4
5
6
7
8
9
10

Sujeto a:

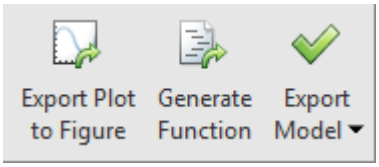

Model 2
Malignant
Benign
True Class
449
9
17
224
Malignant
Benign
Predicted Class



Y también al gráfico ROC:



Asimismo, podemos también estudiar la programación del entrenamiento haciendo:



Se encuentra que el método KNN tiene mayor precisión que el método por regresión logística.

```
function Data = importfile(filename, dataLines)
```

```
    if nargin < 2
        dataLines = [1, Inf];
    end
    opts = delimitedTextImportOptions("NumVariables", 11);
    opts.DataLines = dataLines;
    opts.Delimiter = ",";
    opts.VariableNames = ["ID","radius","texture","perimeter","area","smoothness","compactness"
    opts.VariableTypes = ["double", "double", "double", "double", "double", "double", "double",
    opts.ExtraColumnsRule = "ignore";
    opts.EmptyLineRule = "read";
    opts = setvaropts(opts, ["ID","radius","texture","perimeter","area","smoothness","compactne
    opts = setvaropts(opts, ["ID","radius","texture","perimeter","area","smoothness","compactne
    Data = readtable(filename, opts);
end
```

EJERCICIO 3:

```
clear all
clearvars
close all
clc
```

Reduciremos el problema multiclase a el de uno de dos clases usando la clasificación "One vs All" de la teoría de Support Vector Machine. En primer lugar, hagamos los siguientes cambios:

```
Data = importfile("C:\Users\pc\Desktop\MATLAB_CODES\PROCESAMIENTO DE SEÑALES Y DATOS\PCs\2022-1
Data.Tipo(Data.quality >= 0 & Data.quality <= 2) = "Muy_Mala";
Data.Tipo(Data.quality >= 3 & Data.quality <= 4) = "Mala";
Data.Tipo(Data.quality >= 5 & Data.quality <= 6) = "Media";
Data.Tipo(Data.quality >= 7 & Data.quality <= 8) = "Alta";
Data.Tipo(Data.quality >= 9 & Data.quality <= 10) = "Muy_Alta";
Data.Tipo = categorical(Data.Tipo);
```

Comenzamos con el procedimiento clásico de la separación de data por HoldOut, hacemos:

```
nr = height(Data);
```

Trabajamos con el 80 por ciento de la data, esto es:

```
part = 0.2;
```

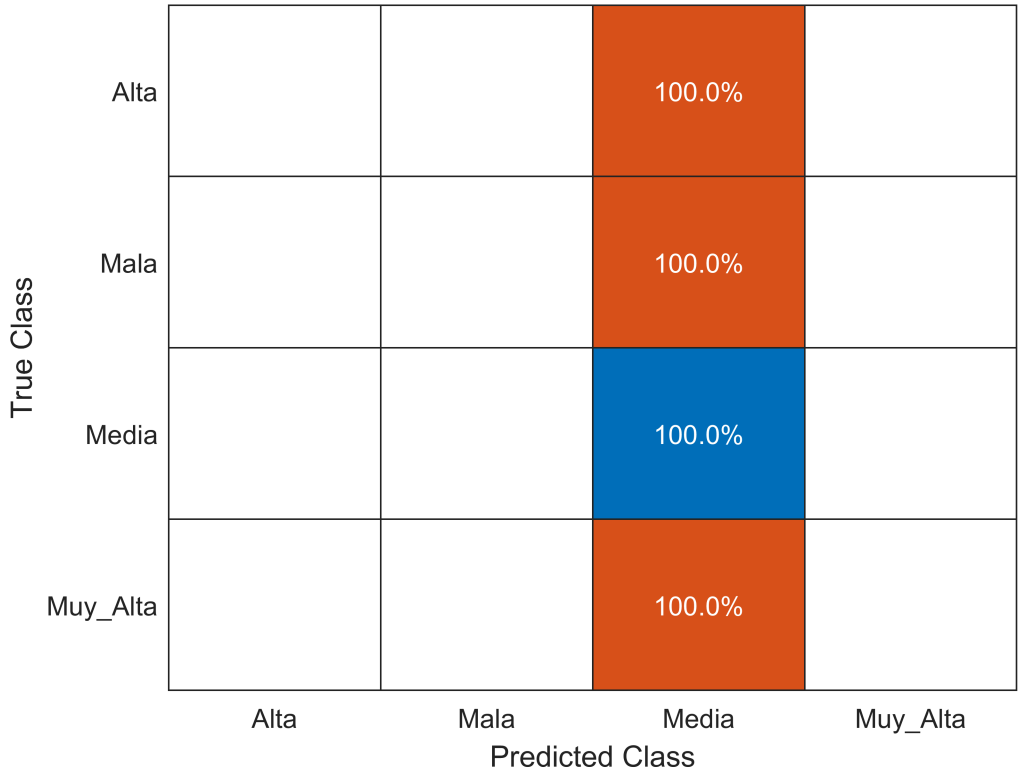
De ahí, hacemos:

```
DataTrain = cvpartition(nr,'HoldOut',part);
iTrain = training(DataTrain);
DataTrainVal = Data(iTrain,:);
```

Y habiendo creado una función para el problema, tenemos:

Primer Function Kernel a usar:

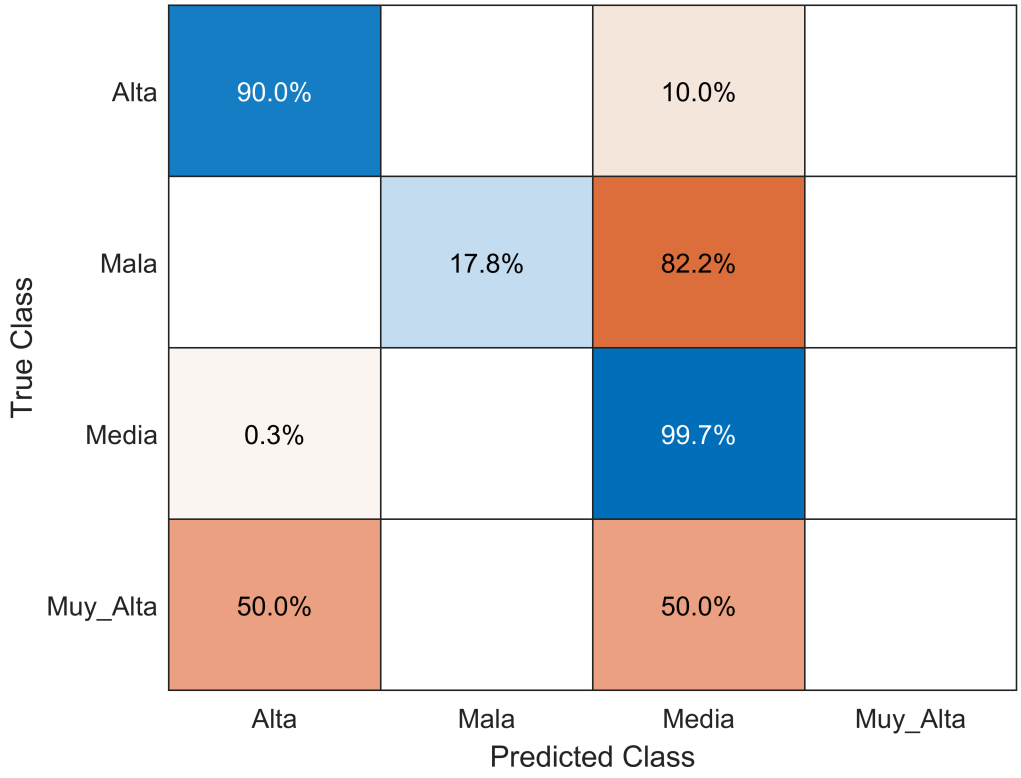
```
[AccuracyL] = SVMmodels(DataTrainVal,"linear")
```




```
AccuracyL = 74.8405
```

Segunda Function Kernel a usar:

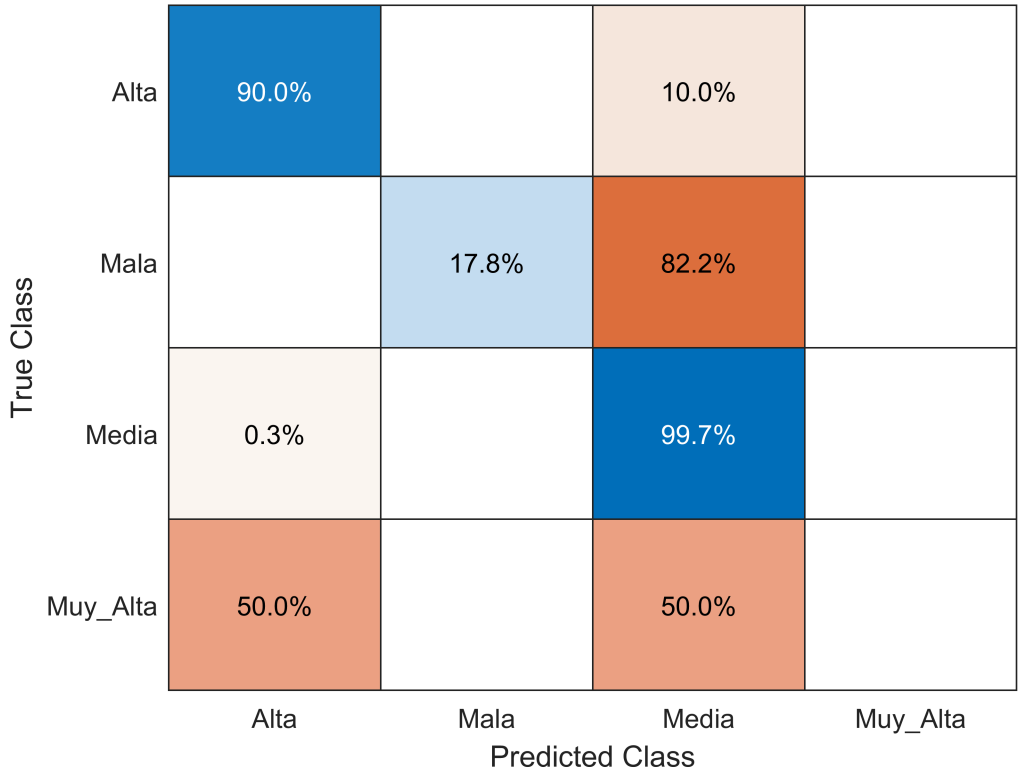
```
[AccuracyG] = SVMmodels(DataTrainVal,"gaussian")
```




```
AccuracyG = 94.4374
```

Tercera Function Kernel a usar:

```
[AccuracyP] = SVMmodels(DataTrainVal,"rbf")
```




```
AccuracyP = 94.4374
```

```
function Data = importfile(filename, dataLines)
    if nargin < 2
        dataLines = [2, Inf];
    end
    opts = delimitedTextImportOptions("NumVariables", 12);
    opts.DataLines = dataLines;
    opts.Delimiter = ",";
    opts.VariableNames = ["fixedAcidity", "volatileAcidity", "citricAcid", "residualSugar", "ch
    opts.VariableTypes = ["double", "double", "double", "double", "double", "double", "double",
    opts.ExtraColumnsRule = "ignore";
    opts.EmptyLineRule = "read";
    Data = readtable(filename, opts);
end
function [Accuracy] = SVMmodels(DataTrainVal,KernelFunction)
    Template = templateSVM('KernelFunction',KernelFunction, 'Standardize', true);
    SVMmodel = fitcecoc(DataTrainVal,"Tipo","PredictorNames",["fixedAcidity", "volatileAcidity'
        "residualSugar", "chlorides", "freeSulfurDioxide", "totalSulfurDioxide", "density", "pH
        "sulphates", "alcohol"], "Learners",Template,"Coding","onevsone");
    trueData = DataTrainVal.Tipo;
    predictSVM = predict(SVMmodel,DataTrainVal);
    Accuracy = sum(trueData==predictSVM)/length(trueData)*100;
    confusionchart(trueData,predictSVM,"Normalization","row-normalized")
end
```

EJERCICIO 4:

```
clear all
clearvars
close all
clc
```

Importamos:

```
load data2.mat
```

Hacemos la tabla correspondiente:

```
Data = array2table([force target]);
clearvars -except Data force
```

Ahora bien, lo que haremos será modificar esta tabla de tal manera de que su última fila esté definida por la designación correspondiente al nombre de la clase referidas en las entradas de target ("OK","OVERLOAD","CRACK")

```
w = width(Data);
Names = [1:w];
VarNames = string(Names);
Data.Properties.VariableNames = VarNames;
```

```
ans = 2000×1    Rows 1991:2000
     1
     1
     1
     1
     1
     1
     1
     2
     1
     1
```

```
Data.Tipo(Data.(num2str(w)) == 1) = "OK";
Data.Tipo(Data.(num2str(w)) == 2) = "OVERLOAD";
Data.Tipo(Data.(num2str(w)) == 3) = "CRACK";
Data.Tipo = categorical(Data.Tipo);
Data = removevars(Data,num2str(w));
```

Ahora bien, hacemos la separación para luego trabajar con el 70 por ciento de la data:

```
nr = height(Data);
```

Esto es:

```
part = 0.3;
DataTrain = cvpartition(nr,'HoldOut',part);
iTrain = training(DataTrain);
DataTrainVal = Data(iTrain,[1:w]);
```

Es claro que nuestras variables de interés ahora serán las correspondientes a las columnas de la matriz force.

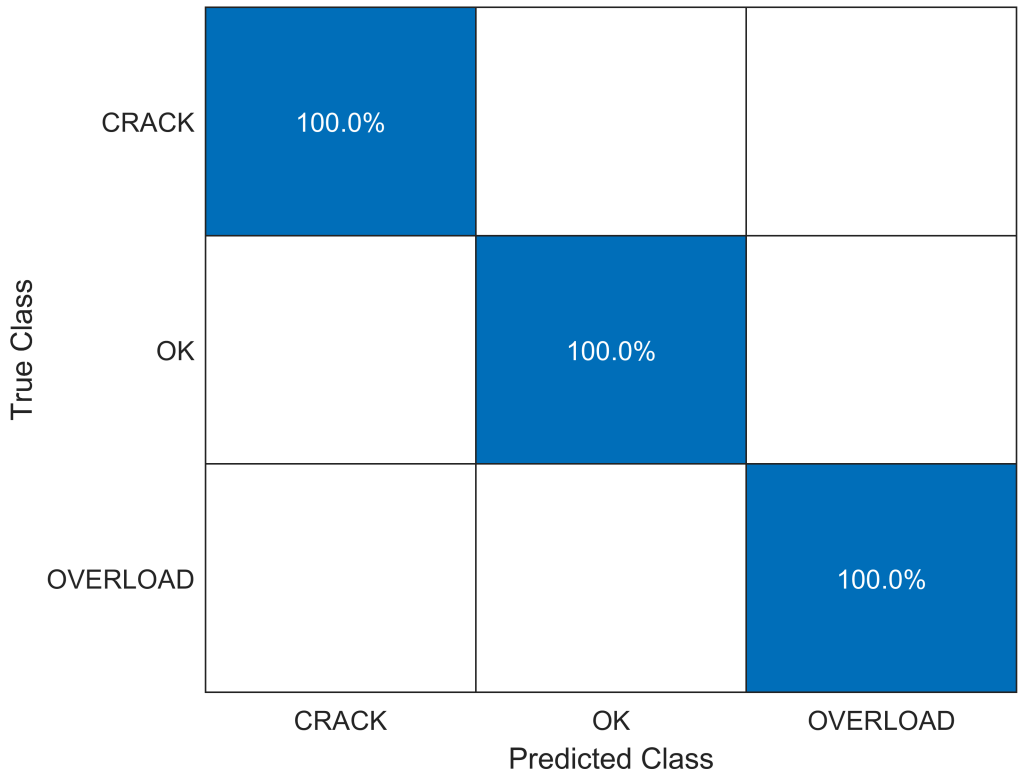
Y teniendo en cuenta que "w" contenía un término más (el vector adicional columna adicional removido, target), trabajaremos con las variables:

```
Names = [1:w-1];
VarNames = string(Names);
```

Clasificamos mediante tres modelos:

MODELO 1:

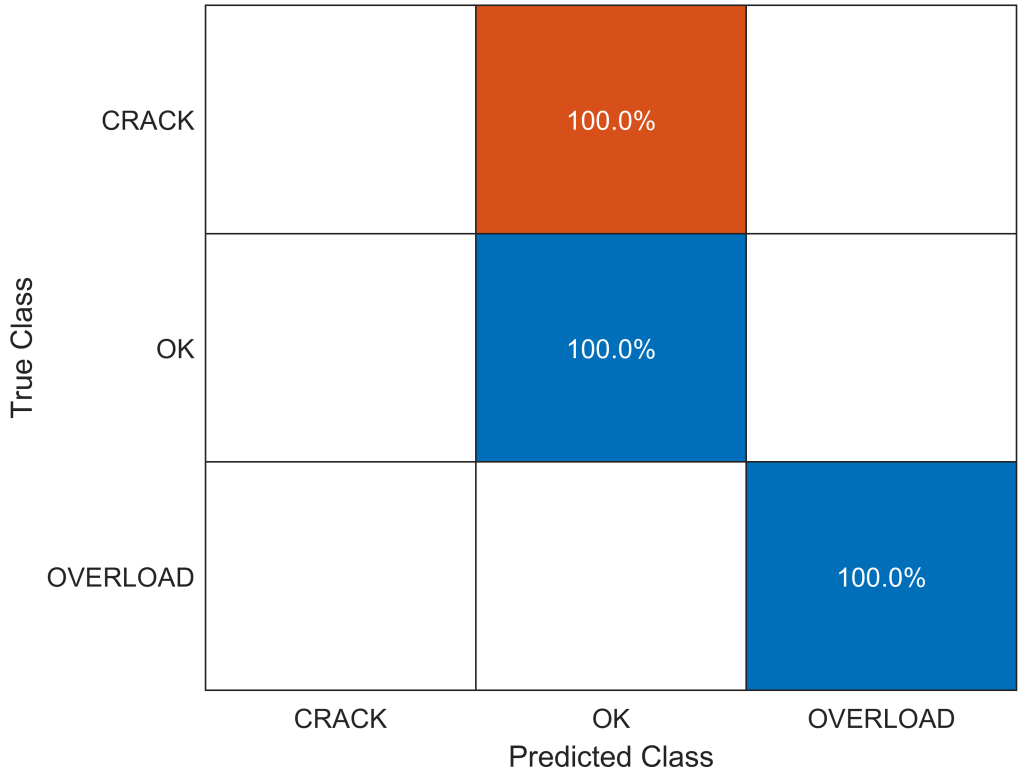
```
[AccuracyL] = SVMmodels(DataTrainVal,"linear",VarNames)
```




```
AccuracyL =
   100
```

MODELO 2:

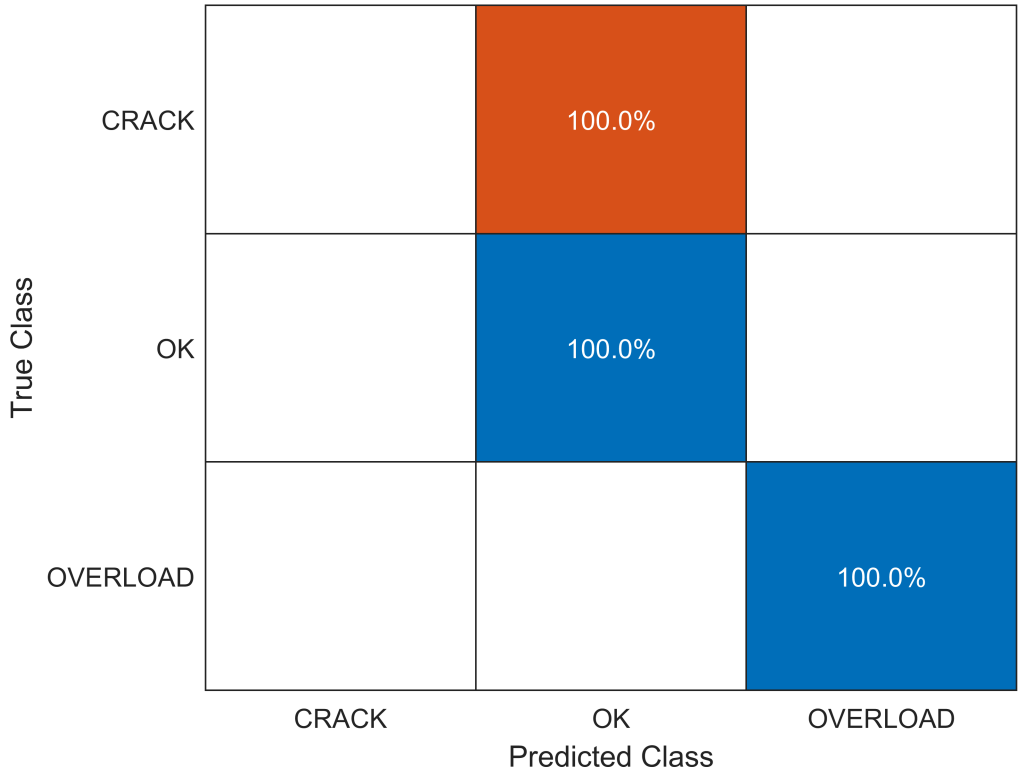
```
[AccuracyG] = SVMmodels(DataTrainVal,"gaussian",VarNames)
```




```
AccuracyG =
  98.857142857142861
```

MODELO 3:

```
[AccuracyP] = SVMmodels(DataTrainVal,"rbf",VarNames)
```




```
AccuracyP =
  98.857142857142861
```

Ahora hacemos uso del PCA:

```
mforce = mean(force);
rforce = range(force);
sforce = (force-mforce)./rforce;
[P,S,V] = pca(sforce);
```

Lo siguiente sería identificar las componentes principales que representan un 95% de la varianza, y luego repetir el cómputo del entrenamiento por SVM con la nueva data correspondiente a las nuevas carácteristicas, así como procedimos anteriormente. Por motivos de tiempo, dejo acá la solución.

```
function [Accuracy] = SVMmodels(DataTrainVal,KernelFunction,VarNames)
    Template = templateSVM('KernelFunction',KernelFunction,'Standardize', true);
    SVMmodel = fitcecoc(DataTrainVal,"Tipo","PredictorNames",VarNames,"Learners",Template,"Codi
    trueData = DataTrainVal.Tipo;
    predictSVM = predict(SVMmodel,DataTrainVal);
    Accuracy = sum(trueData==predictSVM)/length(trueData)*100;
    confusionchart(trueData,predictSVM,"Normalization","row-normalized")
end
```