Registros de votación del Congreso de los Estados Unidos en el año 1984

Clasificar como republicano o demócrata (Clasificación Supervisada)

Méndez Pool Joan de Jesús | 160300102

El objetivo del proyecto es comparar diferentes técnicas para resolver un problema de clasificación binario sobre el conjunto de datos de los *Registros de votación del Congreso de los Estados Unidos en el año 1984*, Este conjunto de datos incluye votos para cada una de las cámaras de *Representantes congresistas* sobre los 16 votos clave identificados por el CQA. La CQA enumera nueve tipos diferentes de votos: *voted for (votados para), paired for (emparejados para), y announced for (anunciado para)* (estos tres son simplificados a la categoría *sí*), *voted agains (votaron en contra), paired against (emparejado en contra) y announced against (anunciado en contra) (estos tres simplificado a la categoría no), voted present (votó presente), voted present to avoid conflict of interest (votó presente para evitar conflictos de interés), y no votó ni dio a conocer una posición* (estos tres simplificados a una disposición *desconocida*).

Los atributos que contiene el conjunto de datos son los siguientes:

1. Class Name: 2 clases (democrat, republican)
2. handicapped-infants: 2 clases (y,n)
3. water-project-cost-sharing: 2 clases (y,n)
4. adoption-of-the-budget-resolution: 2 clases (y,n)
5. physician-fee-freeze: 2 clases (y,n)
6. el-salvador-aid: 2 clases (y,n)
7. religious-groups-in-schools: 2 clases (y,n)
8. anti-satellite-test-ban: 2 clases (y,n)
9. aid-to-nicaraguan-contras: 2 clases (y,n)
10. mx-missile: 2 clases (y,n)
11. immigration: 2 clases (y,n)
12. synfuels-corporation-cutback: 2 clases (y,n)
13. education-spending: 2 clases (y,n)
14. superfund-right-to-sue: 2 clases (y,n)
15. crime: 2 (y,n)
16. duty-free-exports: 2 clases (y,n)
17. export-administration-act-south-africa: 2 clases (y,n)

La variable de respuesta es *"Class Name"*, el resto de las variables son predictoras, observamos que todas las variables son categóricas. Por medio de las técnicas de clasificación conocidas como *K - Nearest Neighbors*, *Logistic Regression*, *Neural Network*, *Support Vector Machine*, *Decision Tree* y *Naive Bayes* debemos predecir si congresista pertene al partido republicano o al partido demócrata de acuerdo a los vatos realizados de sus semejantes sobre las propuestas del congreso del 84'

Antecedentes

Publicaciones anteriores:

```
    1. Schlimmer, J. C. (1987).  Concept acquisition through
       representational adjustment.  Doctoral dissertation, Department of
       Information and Computer Science, University of California, Irvin
  e, CA.
     - Resultados: aproximadamente 90% -95% de precisión parece ser un es
  calonamiento asintótico.
     - Atributo previsto: afiliación partidaria (2 clases).
```

El proyecto fue realizado en *Python* con los módulos de *sklearn* para construir los modelos clasificadores a comparar sobre el conjunto de datos de *house-votes-84.data*.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
# Sklearn library
from sklearn.pipeline import Pipeline
# Proyectar los componentes en un espacio de dos dimensiones
from sklearn.decomposition import PCA
# Procesar el escalamiento de datos
from sklearn.preprocessing import StandardScaler, scale
# Separación de conjunto de entrenamiento y prueba
from sklearn.model_selection import train_test_split
# Métricas
from sklearn.metrics import confusion_matrix, classification_report, roc_a
from sklearn.metrics import accuracy_score, precision_score, recall_score,
# Interación de modelos de entrenamiento
from sklearn.model_selection import GridSearchCV
# K - Vecinos más cercanos
from sklearn.neighbors import KNeighborsClassifier
# Regresión logistica
from sklearn.linear_model import LogisticRegression
# Redes Neuronales
from sklearn.neural_network import MLPClassifier
# Maquina de Soporte Vectorial
from sklearn.svm import SVC
# Arboles de Decision
from sklearn.tree import DecisionTreeClassifier
# Naive Bayes
from sklearn.naive_bayes import GaussianNB

# ROC Curve ( Característica Operativa del Receptor)
from sklearn.metrics import roc_curve

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

```

Marco teórico

K - Vecinos más Cercanos

El K-NN (K-Nearest Neighbor) es un algoritmo de aprendizaje supervisado, el algoritmo trabaja a partir de un conjunto de datos inicial, el objetivo será el de clasificar correctamente todas las instancias nuevas. El conjunto de datos típico de este tipo de algoritmos está formado por varios atributos descriptivos y un solo atributo objetivo (también denominado clase).

El algoritmo clasifica cada dato nuevo en el grupo que corresponda, según tenga k vecinos más cercanos de un grupo o de otro. Se encarga de calcular la distancia de cada elemento nuevo a cada uno de los existentes, y ordena dichas distancias de menor a mayor para ir seleccionando el grupo a clasificar. Este grupo será el de mayor frecuencia con menores distancias.

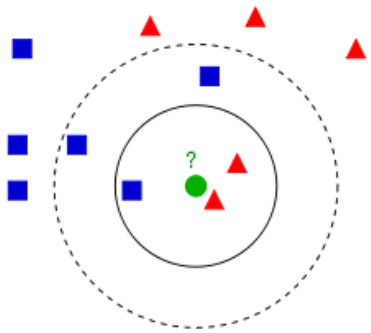


Fig. 1: K-Nearest Neighbor algorithm

Regresión Logística

La regresión logística es una técnica de aprendizaje supervisado para clasificación binaria, aunque se puede adaptar a problemas de clasificaciín multiclase.

Se supone que se dispone de un conjunto de datos, en los cuales la etiqueta $y_i \in \{-1, 1\}$:

$$\begin{bmatrix} x_{10} & x_{11} & x_{12} & \cdots & x_{1d} & y_1 \\ x_{10} & x_{21} & x_{22} & \cdots & x_{2d} & y_2 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ x_{N0} & x_{N1} & x_{N2} & \cdots & x_{Nd} & y_N \end{bmatrix}$$

En la regresión logística se ajusta un modelo lineal a la razón entre las dos probabilidades de clase:

$$\ln\left[\frac{P(Y=1|X=x)}{P(Y=-1|X=x)}\right] = \alpha^T x$$

Siendo $\alpha = [\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_d]^T$ el vector de coeficientes del modelo lineal

Observación

En la regresión logística no se establece la condición de que las observaciones provengan de distribuciones gaussianas multivariadas.

Función Sigmoide

Se puede despejar la probabilidad $P(Y = 1|X = x)$ del modelo lineal:

$$P(Y = 1|X = x) = \frac{e^{\alpha^T x}}{1 + e^{\alpha^T x}} = \sigma(\alpha^T x)$$

La función $\sigma()$ es una función **sigmoide**.

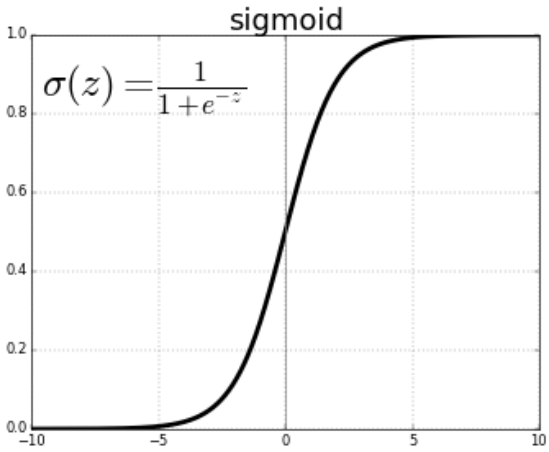


Fig. 2: Sigmoid Function

Redes Neuronales

Perceptrón

Un perceptrón es un clasificador que permite ajustar regiones lineales para separar observaciones de diferentes clases.

El modelo del perceptrón es el siguiente:

$$y = s(\alpha_0 + \alpha_1 x_1 + \cdots + \alpha_d x_d)$$

La siguiente imagen es una representación gráfica del perceptrón

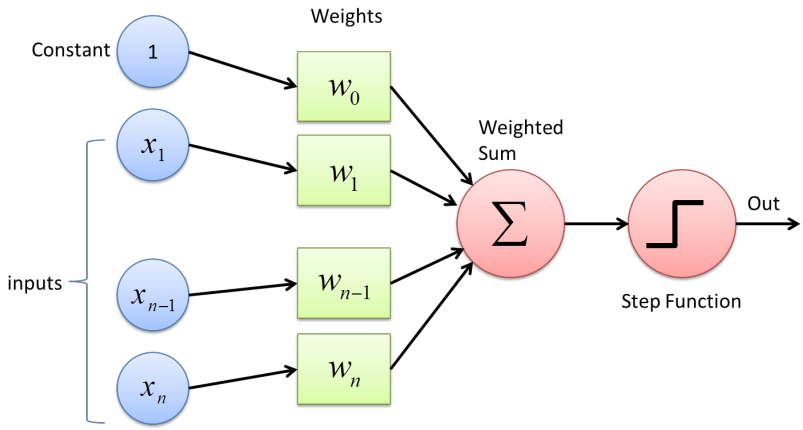


Fig. 3: Perceptron

Los coeficientes (o pesos) deben determinarse minimizando la suma de errores al cuadrado:

$$E(\alpha_0, \alpha_1, \ldots, \alpha_d) = \sum_i [y_i - s(\alpha_0 + \alpha_1 x_{i1} + \cdots + \alpha_d x_{id})]^2$$

Sin embargo, dado que la función signo, $s(x)$, no es derivable, por lo que para encontrar los pesos que minimicen la suma de errores es necesario utilizar un algoritmo iterativo, inspirado en el algoritmo de **descenso de gradiente**, que es muy sensible a los óptimos locales, por lo que su aplicación se dificulta en problemas de clasificación de grandes dimensiones (vectores de características con muchas componentes).

Perceptrón (salida derivable)

Dado que no se puede derivar la función signo se limita la aplicación de algún método de optimización sofisticado, pues estos metodos generalmente se basan en la evaluación y aplicación del gradiente de la función a minimizar.

Una primera adecuación al perceptrón consiste entonces en colocar a la salida una función derivable que aproxime a la función signo. A este tipo de funciones se les denomina **sigmoides**.

Multiperceptrones

Un perceptrón permite *aprender* funciones lógicas básicas como las siguientes:

- **OR**. Considera los siguientes pesos: $\alpha_0 = -1, \alpha_1 = 1, \alpha_2 = 1$
- **AND**. Con
$$\alpha_0 = -2, \alpha_1 = 1, \alpha_2 = 1$$
- **NAND**. Con
$$\alpha_0 = 2, \alpha_1 = -1, \alpha_2 = -1$$
- ¿**XOR**?

Un perceptrón no puede utilizarse para aprender la función **XOR**, sin embargo, sí es posible hacerlo por medio de dos perceptrones: **XOR = (OR) AND (NAND)**

Red Neuronal

Una red neuronal es un conjunto de capas de perceptrones conectadas en serie:

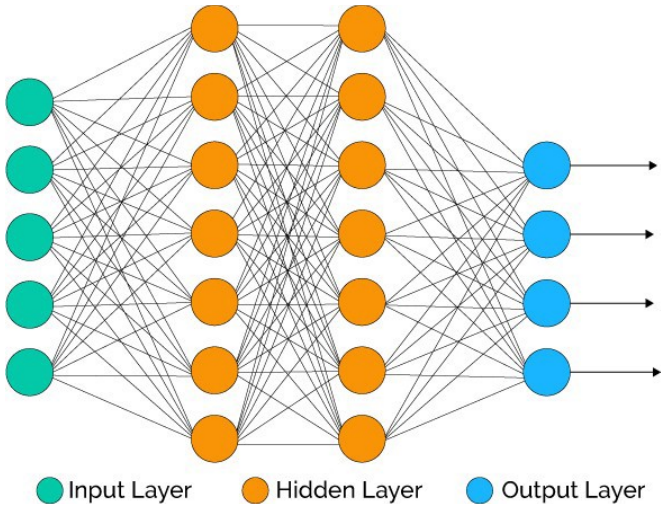


Fig. 4: Neural Network

La red neuronal permite extender las aplicaciones de los perceptrones simples considerando lo siguiente:

- A los elementos de la capa de entrada se conectan las componentes de un vector de características.
- Las capas intermedias permiten realizar transformaciones no-lineales muy generales a las variables originales.
- El tener múltiples perceptrones en la capa de salida permite la aplicación de las redes neuronales en problemas de clasificación con un número arbitrario de etiquetas.
- El entrenamiento de la red neuronal se realiza mediante la minimización de la suma de erroes al cuadrado utilizando algún método basado en gradiente.
- Si la optimizaciónse realiza por medio de descenso de gradiente, entonces al algoritmo se le llama **retropropagación** (backpropagation).

Máquinas de soporte Vectorial

Máquinas de soporte Vectorial (Suporter Vector Machine en inglés), es una técnica conocida técnica moderna y efectiva del campo de la Inteligencia Artificial, aplicada fundamentalmente al procesamiento de grandes cantidades de información.

Las Máquinas de Soporte Vectorial (MSV) han tenido un formidable desarrollo en los últimos años.

Se presentarán los fundamentos teóricos que definen estos sistemas de aprendizaje:

- Uno de los conceptos fundamentales en esta técnica es el algoritmo Vector de Soporte (VS) es una generalización no-lineal del algoritmo Semblanza Generalizada, desarrollado en la Rusia en los años sesenta. El desarrollo de los VS trae consigo el surgimiento de las Máquinas de Soporte Vectorial. Estas son sistemas de aprendizaje que usan un espacio de hipótesis de funciones lineales en un espacio de rasgos de mayor dimensión, entrenadas por un algoritmo proveniente de la teoría de optimización.

- La Minimización del Riesgo Empírico y la [Dimensión de Vapnik-Chervonenkis] son fundamentales en las Máquinas de Soporte Vectorial. Dicho de manera más sencilla el algoritmo se enfoca en el problema general de aprender a discriminar entre miembro positivos y negativos de una clase de vectores de n-dimensional dada.

- Las MSV pertenecen a la familia de clasificadores lineales. Mediante una función matemática denominada kernel, los datos originales se redimensionan para buscar una separabilidad lineal de los mismos. Una característica de las MSV es que realiza un mapeo de los vectores de entrada para determinar la linealidad o no de los casos los cuales serán integrados a los Multiplicadores de Lagrange para minimizar el Riesgo Empírico y la Dimensión de Vapnik-Chervonenkis. De manera general, las Máquinas de Soporte Vectorial permiten encontrar un hiperplano óptimo que separe las clases.

Kernel

Las funciones kernel son funciones matemáticas que se emplean en las Máquinas de Soporte Vectorial. Estas funciones son las que le permiten convertir lo que sería un problema de clasificación no-lineal en el espacio dimensional original, a un sencillo problema de clasificación lineal en un espacio dimensional mayor.

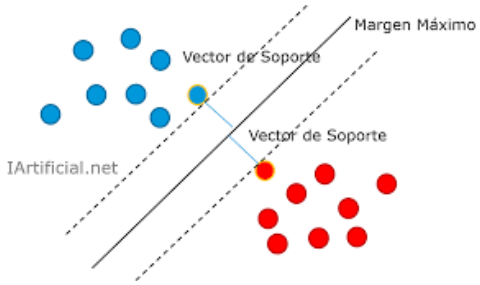


Fig. 5: Support Vector Machine

Árboles de Decisión

Classification And Regression Tree (CART) son un tipo de algoritmos de aprendizaje supervisado (tiene como objetivo una variable predefinida). Son principalmente usados en problemas de clasificación, las variables de entrada y salida pueden ser variables categóricas o variables continuas. El algoritmo divide el espacio de predictores (variables independientes) en regiones distintas y no sobrepuestas. Este enfoque fue desarrollado por Breiman et al. (1984).

Generalizando:

$$SSS = \sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y_{R_m}})^2$$

Se divide la población o muestra en conjuntos homegéneos basados en la variable de entrada más significativa. La construcción del árbol sigue un enfoque de división binaria recursiva (top-down greddy approach). Greedy -> analiza la mejor variable para ramificación sólo en el proceso de división actual.

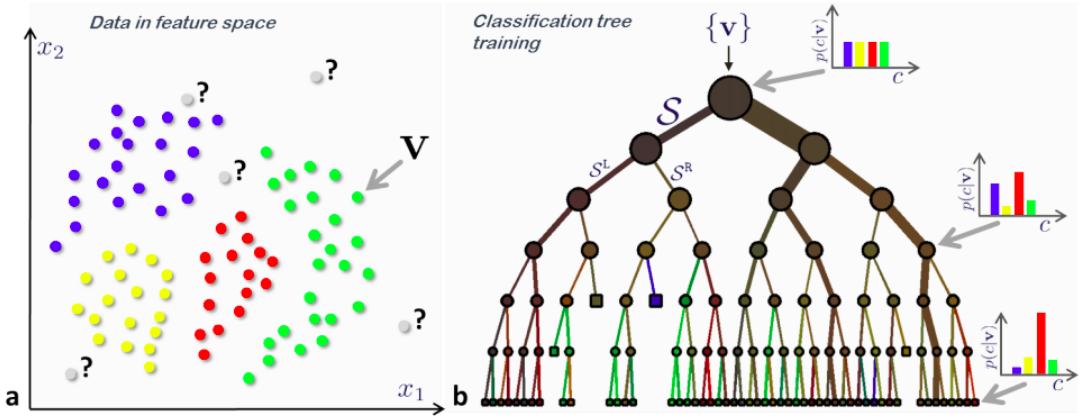


Fig. 6: Classification And Regression Tree

Teorema de Naive Bayes

Un clasificador Naive Bayes es un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras adicionales. Es a causa de estas simplificaciones, que se suelen resumir en la hipótesis de independencia entre las variables predictoras, que recibe el apelativo de naive, es decir, ingenuo.

El Clasificador de Naive Bayes asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de cualquier otra de las características, dada la clase variable.

Teorema de Bayes:

$$P(y \mid X) = \frac{P(X \mid y)P(y)}{P(X)}$$

Usando el teorema de Bayes, podemos encontrar la probabilidad de que ocurra y, dado que X ha ocurrido. Observamos que X es la evidencia e y es la hipótesis. La suposición hecha aquí es que los predictores son independientes. Es decir, la presencia de una característica particular no afecta a la otra. Por eso se llama ingenuo.

Tipos de clasificador ingenuo de Bayes:

Bayes ingenuos multinomiales:

Esto se usa principalmente para problemas de clasificación de documentos, es decir, si un documento pertenece a la categoría de deportes, política, tecnología, etc. Las características utilizadas por el clasificador son la frecuencia de las palabras presentes en el documento.

Bernoulli ingenuo Bayes:

Esto es similar a las bayes ingenuas multinomiales, pero los predictores son variables booleanas. Los parámetros que usamos para predecir la variable de clase solo toman valores sí o no, por ejemplo, si una palabra aparece en el texto o no.

Bayes ingenuos gaussianos:

Cuando los predictores toman un valor continuo y no son discretos, suponemos que estos valores se muestrean a partir de una distribución gaussiana.

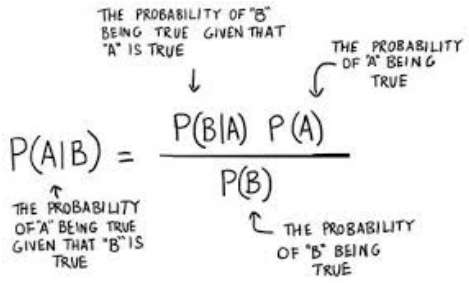


Fig. 7: Naive Bayes Theorem

Carga del Conjunto Datos

Cargamos el conjunto de datos de *house-votes-84.data* para visualizar cuales son nuestras variables predictoras y hacer una observación sobre la variable categórica.

In [2]:
```
1 data = pd.read_csv("house-votes-84.data", header=None)
2 colsname = [ line.replace("\n", "") for line in open("house-votes-84-heade
3 data.columns = colsname
4 print("Dataframe shape: {} rows x {} columns".format(data.shape[0], data.s
5 data.head(10)
6
```

```
Dataframe shape: 435 rows x 17 columns
```

Out[2]:

	Class Name	handicapped-infants	water-project-cost-sharing	adoption-of-the-budget-resolution	physician-fee-freeze	el-salvador-aid	religious-groups-in-schools	anti-satellite-test-ban	aid-to-nicaraguan-contras	mi missi
0	republican	n	y	n	y	y	y	n	n	
1	republican	n	y	n	y	y	y	n	n	
2	democrat	?	y	y	?	y	y	n	n	
3	democrat	n	y	y	n	?	y	n	n	
4	democrat	y	y	y	n	y	y	n	n	
5	democrat	n	y	y	n	y	y	n	n	
6	democrat	n	y	n	y	y	y	n	n	
7	republican	n	y	n	y	y	y	n	n	
8	republican	n	y	n	y	y	y	n	n	
9	democrat	y	y	y	n	n	n	y	y	

Ahora procedemos a observar la información de nuestro conjunto de datos, como por ejemplo el tipo de dato, el número de entradas, el total de columnas, y si el conjunto de datos posee datos nulos.

In [3]:
```
1 data.info()
2
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 17 columns):
Class Name                                435 non-null object
handicapped-infants                       435 non-null object
water-project-cost-sharing                435 non-null object
adoption-of-the-budget-resolution         435 non-null object
physician-fee-freeze                      435 non-null object
el-salvador-aid                           435 non-null object
religious-groups-in-schools               435 non-null object
anti-satellite-test-ban                   435 non-null object
aid-to-nicaraguan-contras                 435 non-null object
mx-missile                                435 non-null object
immigration                               435 non-null object
synfuels-corporation-cutback              435 non-null object
education-spending                        435 non-null object
superfund-right-to-sue                    435 non-null object
crime                                     435 non-null object
duty-free-exports                         435 non-null object
export-administration-act-south-africa    435 non-null object
dtypes: object(17)
memory usage: 57.9+ KB
```

Descripción de las características

Realizamos un resumen de las para conocer sus medidas estadísticas, ya que las variables son categóricas obtenemos solamente la cantidad de datos, cuantos valores únicos posee, el valor que más se repite en cada columna y la frecuencia de dicho término.

In [4]:
```
1 data.describe()
2
```

Out[4]:

	Class Name	handicapped-infants	water-project-cost-sharing	adoption-of-the-budget-resolution	physician-fee-freeze	el-salvador-aid	religious-groups-in-schools	anti-satellite-test-ban	aid-to-nicaraguan-contras	r
count	435	435	435	435	435	435	435	435	435	
unique	2	3	3	3	3	3	3	3	3	
top	democrat	n	y	y	n	y	y	y	y	
freq	267	236	195	253	247	212	272	239	242	

Procedemos a eliminar los valores no conocidos en este caso están denominados por la etiqueta *'?'*, remplazamos lo datos con la constante de valores no asignados ("NA"). Por lo que lo siguiente es eliminar todas las observaciones con campos no asignados una vez aplicado el filtro.

In [5]:
```
1 data = data.replace("?", np.nan)
2 data = data.dropna()
3 print("New Dataframe shape: {} rows x {} columns".format(data.shape[0], da
4 data.info()
5
```

```
New Dataframe shape: 232 rows x 17 columns
<class 'pandas.core.frame.DataFrame'>
Int64Index: 232 entries, 5 to 431
Data columns (total 17 columns):
Class Name                                  232 non-null object
handicapped-infants                         232 non-null object
water-project-cost-sharing                  232 non-null object
adoption-of-the-budget-resolution           232 non-null object
physician-fee-freeze                        232 non-null object
el-salvador-aid                             232 non-null object
religious-groups-in-schools                 232 non-null object
anti-satellite-test-ban                     232 non-null object
aid-to-nicaraguan-contras                   232 non-null object
mx-missile                                  232 non-null object
immigration                                 232 non-null object
synfuels-corporation-cutback                232 non-null object
education-spending                          232 non-null object
superfund-right-to-sue                      232 non-null object
crime                                       232 non-null object
duty-free-exports                           232 non-null object
export-administration-act-south-africa      232 non-null object
dtypes: object(17)
memory usage: 32.6+ KB
```

Ahora observamos que nos quedamos solo con 232 observaciones, los cuales están listos para aplicar los metodos de clasificación. Ahora describimos la información para observar si existe un cambio significativo en la frecuencia de cada termino para cada columna.

In [6]:
```
1 data.describe()
2
```

Out[6]:

	Class Name	handicapped-infants	water-project-cost-sharing	adoption-of-the-budget-resolution	physician-fee-freeze	el-salvador-aid	religious-groups-in-schools	anti-satellite-test-ban	aid-to-nicaraguan-contras	r
count	232	232	232	232	232	232	232	232	232	
unique	2	2	2	2	2	2	2	2	2	
top	democrat	n	n	y	n	y	y	y	y	
freq	124	136	125	123	119	128	149	124	119	

Aún no se puede aplicar por completo los modelos de clasificación ya que los algoritmos no trabajan con valores de tipo *character* por lo que se debe realizar un procesamiento, para realizar está tarea se usa el método *get_dummies*, la función convierte las columnas con valores categóricos en indicadores binarios, donde cada combinación de secuencias binarias representa una clase.

In [7]:
```
data_dummies = pd.get_dummies(data, drop_first=True)
data_dummies.head()
```

Out[7]:

	Class Name_republican	handicapped-infants_y	water-project-cost-sharing_y	adoption-of-the-budget-resolution_y	physician-fee-freeze_y	el-salvador-aid_y	religious-groups-in-schools_y	anti-satellite-test-ban_y	nicar cor
5	0	0	1	1	0	1	1	0	
8	1	0	1	0	1	1	1	0	
19	0	1	1	1	0	0	0	1	
23	0	1	1	1	0	0	0	1	
25	0	1	0	1	0	0	0	1	

Los algoritmos de clasificación supervisada del módulo *scikit-learn* reciben dos entradas, el vector de características y el vector de etiquetas, por lo que debemos separar estas en dos variables que serán conocidas como Y para la variable categórica *Class Name* y X para nuestras variables predictoras.

In [8]:
```
# Dataframe dummie copy
Y = data_dummies["Class Name_republican"]
data_dummies = data_dummies.drop(['Class Name_republican'], axis=1)
X = data_dummies.copy(deep=True)
X.describe()
```

Out[8]:

	handicapped-infants_y	water-project-cost-sharing_y	adoption-of-the-budget-resolution_y	physician-fee-freeze_y	el-salvador-aid_y	religious-groups-in-schools_y	anti-satellite-test-ban_y	aid-to-nicaraguan-contras_y
count	232.000000	232.000000	232.000000	232.000000	232.000000	232.000000	232.000000	232.000000
mean	0.413793	0.461207	0.530172	0.487069	0.551724	0.642241	0.534483	0.512931
std	0.493577	0.499571	0.500168	0.500913	0.498393	0.480377	0.499888	0.500913
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Separar los datos en conjuntos de entrenamiento y prueba

A continuación realizamos la separación del conjunto de datos en dos subconjuntos, el conjunto de entrenamiento se encarga de la construcción de los clasificadores, y el conjunto de prueba es para evaluar la precisión de nuestro modelo de clasificación. Nuestro conjunto de entrenamiento contiene 75% de la información y el conjunto de prueba 25% de la información, con una semilla de valor 124 para siempre obtener los mismos resultados en la separación de los datos.

In [9]:
```
# Separación de conjuntos
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,
print("Cantidad de datos en el conjunto de Entrenamiento: {}".format(X_tra
print("Cantidad de datos en el conjunto de Prueba: {}".format(X_test.count

```

```
Cantidad de datos en el conjunto de Entrenamiento: 174
Cantidad de datos en el conjunto de Prueba: 58
```

K - Nearest Neighbors

Aplicamos el método de vecinos más cercanos por medio del clasificador *KNeighborsClassifier()* obtenido de los módulos de *sklearn*, construimos un *Pipeline*, primero con el comando de escalamiento de datos para posteriormente aplicar el clasificador, definimos nuestros parámetros del clasificador para realizar una búsqueda exhaustiva con validación cruzada, con un valor de $k = 10$, para realizar la búsqueda utilizamos el comando *GridSearchCV()* proveniente de los módulos de *sklearn* para realizar un algoritmo de validación cruzada con el fin de obtener los mejores parámetros para nuestro modelo de clasificación.

Después de realizar la validación cruzada de parámetros debemos entrenar nuestro modelo con el conjunto de datos de entrenamiento.

In [10]:
```
# Pipeline
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
# Hyperparameters
parameters = {'knn__n_neighbors': list(range(1,8)),
              'knn__weights': ["uniform", "distance"]}

# Cross Validation
knn_cv = GridSearchCV(pipeline, parameters, cv=10)
# Fit Model
knn_cv.fit(X_train, y_train)

```

Out[10]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('knn',
                                        KNeighborsClassifier(algorithm='auto
',
                                                             leaf_size=30,
                                                             metric='minkows
ki',
                                                             metric_params=N
one,
                                                             n_jobs=None,
                                                             n_neighbors=5,
p=2,
                                                             weights='unifor
m'))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7],
                         'knn__weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [11]:
```
print("Mejores Parámetros del modelo: {}".format(knn_cv.best_params_))

```

```
Mejores Parámetros del modelo: {'knn__n_neighbors': 4, 'knn__weights': 'unif
orm'}
```

Función para visualizar la matriz de confusión.

Métricas de Evaluación de Clasificación (Confusion Matrix, Accuracy, Precision, Recall, F1, ROC y AUC)

- Una matriz de confusión es una tabla que describe el desempeño de un modelo de clasificación en un conjunto de datos de prueba cuyos valores verdaderos son conocidos. Una matriz de confusión es altamente interpretativa y puede ser usada para estimar un número de otras métricas.

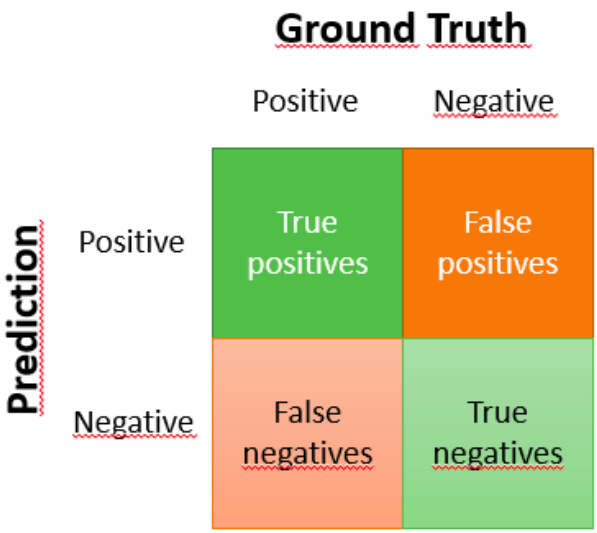


Fig. 8: Confusion Matrix

- La exactitud de la clasificación es la relación entre las predicciones correctas y el número total de predicciones. O más simplemente, con qué frecuencia es correcto el clasificador.
  La fórmula de la exactitud esta definida por:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- La precisión es la relación entre las predicciones correctas y el número total de predicciones correctas previstas. Esto mide la precisión del clasificador a la hora de predecir casos positivos.
  La fórmula de la precisión esta definida por:

$$Precision = \frac{TP}{TP + FN}$$

- La sensibilidad también es llamada en inglés recall, es la relación entre las predicciones positivas correctas y el número total de predicciones positivas. O más simplemente, cuán sensible es el clasificador para detectar instancias positivas. Esto también se conoce como la tasa verdadera positiva.
  La fórmula de la sensibilidad esta definida por:

$$Recall = \frac{TP}{TP + FP}$$

- El puntaje F1 es la medida armónica de la memoria y la precisión, con una puntuación más alta como mejor modelo.
  La fórmula de la sensibilidad esta definida por:

$$Recall = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Una curva ROC (curva de característica operativa del recepto) es un gráfico que muestra el rendimiento de un modelo de clasificación en todos los umbrales de clasificación. Esta curva representa dos parámetros:
  - Tasa de verdaderos positivos.
  - Tasa de falsos positivos.

- AUC significa "área bajo la curva ROC". Esto significa que el AUC mide toda el área bidimensional por debajo de la curva ROC completa. El AUC proporciona una medición agregada del rendimiento en todos los umbrales de clasificación posibles. Una forma de interpretar el AUC es como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio.

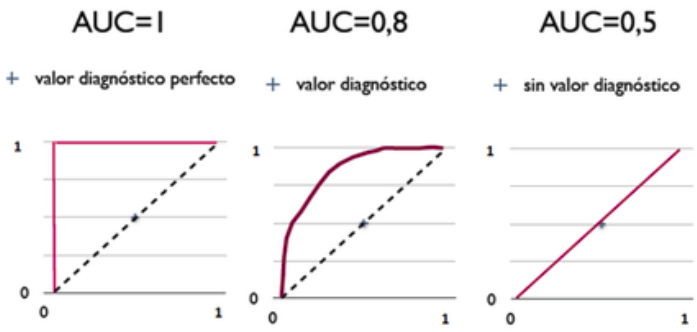


Fig. 9: ROC Curves

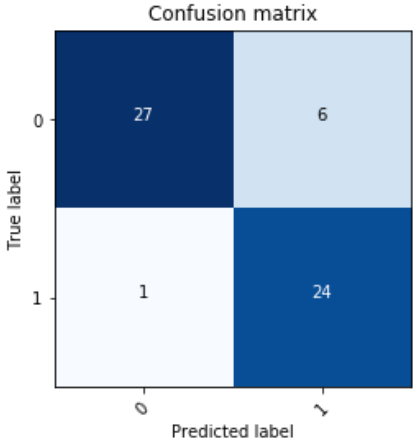
In [12]:

```
# funcion optenida de: https://scikit-learn.org/stable/auto_examples/mode
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion r
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    #plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

def Plot_ROC_Curve(y_test, y_pred_prob):
    # Generate ROC curve values: fpr, tpr, thresholds
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    # Plot ROC curve
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr,  marker='.')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show()

```

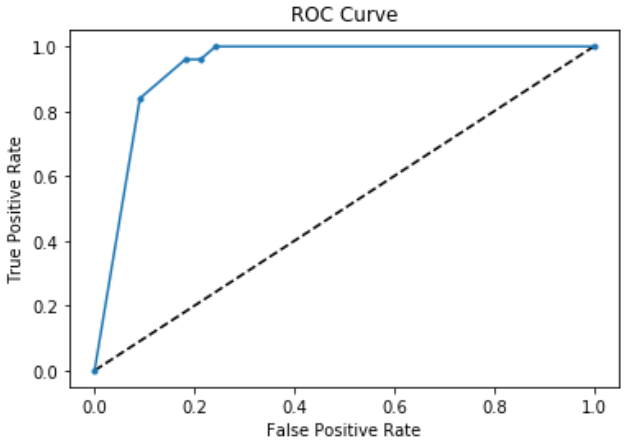
In [13]:

```python
def Classification_Metrics(model, X_test, y_test, labels, show=True):
    # Predicted classes
    y_pred = model.predict(X_test)
    # Compute predicted probabilities: y_pred_prob
    y_pred_prob = model.predict_proba(X_test)[:,1]

    ###### Compute

    # Compute Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    # Calculate metrics
    accu = accuracy_score(y_test, y_pred)
    pre = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    # AUC
    auc = roc_auc_score(y_test, y_pred_prob)
    if show:
        # Confusion Matrix
        plot_confusion_matrix(cm, labels)
        # Print Metrics
        print("Métricas del modelo: \n")
        # Show in percent
        per = 100
        print("Exactitud: {:.2f}%".format(per*accu))
        print("Precisión: {:.2f}%".format(per*pre))
        print("Sensibilidad: {:.2f}%".format(per*rec))
        print("Puntaje F1: {:.2f}%".format(per*f1))

        print("\nAlternativa con Classification_Report: \n")
        # Compute Classification Report
        print(classification_report(y_test, y_pred))
        # Plot ROC Curve
        Plot_ROC_Curve(y_test, y_pred_prob)
        print("AUC: {:.2f}%".format(per*auc))
    # Métricas a retornar
    dic = {"accuracy": accu, "precision": pre, "recall": rec, "f1": f1, "A
    return (dic)

```

In [14]:
```
knn = Classification_Metrics(knn_cv, X_test, y_test, y_test.unique())

```




```
Métricas del modelo:

Exactitud: 87.93%
Precisión: 80.00%
Sensibilidad: 96.00%
Puntaje F1: 87.27%

Alternativa con Classification_Report:

              precision    recall  f1-score   support

           0       0.96      0.82      0.89        33
           1       0.80      0.96      0.87        25

    accuracy                           0.88        58
   macro avg       0.88      0.89      0.88        58
weighted avg       0.89      0.88      0.88        58
```




```
AUC: 93.64%
```

Logistic Regression

Aplicamos el método de Regresión Logística por medio del clasificador *LogisticRegression()* obtenido de los módulos de *sklearn*, construimos un *Pipeline*, primero con el comando de escalamiento de datos para posteriormente aplicar el clasificador, definimos nuestros parámetros del clasificador para realizar una búsqueda exhaustiva con validación cruzada, con un valor de $k = 10$, para realizar la búsqueda utilizamos el comando *GridSearchCV()* para encontrar los mejores parámetros para nuestro modelo de clasificación de regresión logística.

Después de realizar la validación de parámetros debemos entrenar nuestro modelo con el conjunto de datos de entrenamiento.

In [15]:
```
# Pipeline
steps = [('scaler', StandardScaler()),
         ('logreg', LogisticRegression())]
pipeline = Pipeline(steps)
# Hyperparameters
parameters = {'logreg__C': np.logspace(-3,3,7),
              'logreg__penalty': ["l1","l2"]}
# Cross Validation
logreg_cv = GridSearchCV(pipeline, parameters, cv=10)
# Fit model
logreg_cv.fit(X_train, y_train)

```

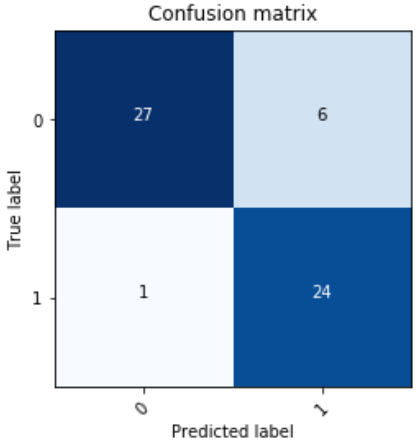
Out[15]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('logreg',
                                        LogisticRegression(C=1.0,
                                                           class_weight=Non
e,
                                                           dual=False,
                                                           fit_intercept=Tru
e,
                                                           intercept_scaling
=1,
                                                           l1_ratio=None,
                                                           max_iter=100,
                                                           multi_class='auto
',
                                                           n_jobs=None,
                                                           penalty='l2',
                                                           random_state=Non
e,
                                                           solver='lbfgs',
                                                           tol=0.0001,
                                                           verbose=0,
                                                           warm_start=Fals
e))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'logreg__C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00,
1.e+01, 1.e+02, 1.e+03]),
                         'logreg__penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [16]:
```
print("Mejores Parámetros del modelo: {}".format(logreg_cv.best_params_))

```

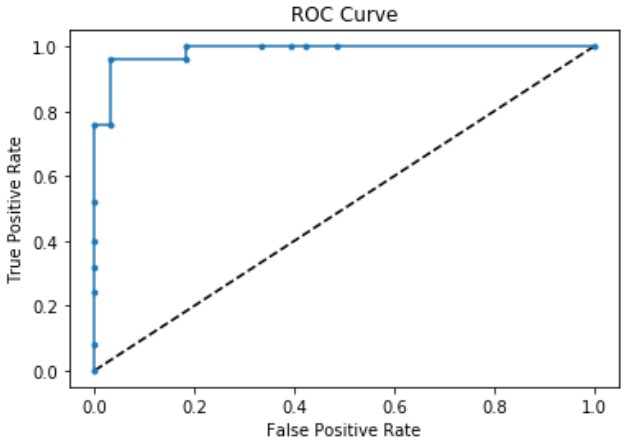
```
Mejores Parámetros del modelo: {'logreg__C': 0.1, 'logreg__penalty': 'l2'}
```

Métricas de Evaluación de Clasificación (Confusion Matrix, Accuracy, Precision, Recall, F1, ROC y AUC)

In [17]:
```
1 logreg = Classification_Metrics(logreg_cv, X_test, y_test, y_test.unique()
2
```




```
Métricas del modelo:

Exactitud: 87.93%
Precisión: 80.00%
Sensibilidad: 96.00%
Puntaje F1: 87.27%

Alternativa con Classification_Report:

              precision    recall  f1-score   support

           0       0.96      0.82      0.89        33
           1       0.80      0.96      0.87        25

    accuracy                           0.88        58
   macro avg       0.88      0.89      0.88        58
weighted avg       0.89      0.88      0.88        58
```




```
AUC: 98.67%
```

Neural Network

Aplicamos el método de Redes Neuronales por medio del clasificador *MLPClassifier()* obtenido de los módulos de *sklearn*, construimos un *Pipeline*, primero con el comando de escalamiento de datos para posteriormente aplicar el clasificador, definimos nuestros parámetros del clasificador para realizar una búsqueda exhaustiva con validación cruzada, con un valor de $k = 10$, para realizar la búsqueda utilizamos el comando *GridSearchCV()* para encontrar los mejores parámetros para nuestro modelo de clasificación.

Después de realizar la validación de parámetros debemos entrenar nuestro modelo con el conjunto de datos de entrenamiento.

In [18]:
```
# Pipeline
steps = [('scaler', StandardScaler()),
         ('MLPC',  MLPClassifier())]
pipeline = Pipeline(steps)
# Hyperparameters
parameters = {'MLPC__activation': ['logistic', 'tanh'],
              'MLPC__solver': ['lbfgs'],
              'MLPC__alpha': 10.0 ** -np.arange(1, 10),
              'MLPC__hidden_layer_sizes':np.arange(10, 15)}
# Cross Validation
MLPC_cv = GridSearchCV(pipeline, parameters, cv=10)
# Fit model
MLPC_cv.fit(X_train, y_train)

```

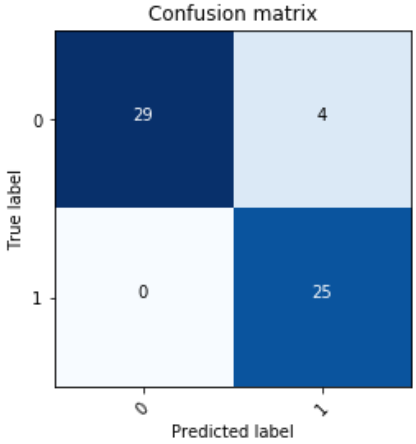
Out[18]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('MLPC',
                                        MLPClassifier(activation='relu',
                                                      alpha=0.0001,
                                                      batch_size='auto',
                                                      beta_1=0.9, beta_2=0.9
99,
                                                      early_stopping=False,
                                                      epsilon=1e-08,
                                                      hidden_layer_sizes=(10
0,),
                                                      learning_rate='constan
t',
                                                      learning_rate_init=0.0
01,
                                                      max_f...
                                                      warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'MLPC__activation': ['logistic', 'tanh'],
                         'MLPC__alpha': array([1.e-01, 1.e-02, 1.e-03, 1.e-0
4, 1.e-05, 1.e-06, 1.e-07, 1.e-08,
       1.e-09]),
                         'MLPC__hidden_layer_sizes': array([10, 11, 12, 13,
14]),
                         'MLPC__solver': ['lbfgs']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [19]:
```
1 print("Mejores Parámetros del modelo: {}".format(MLPC_cv.best_params_))
2
```

```
Mejores Parámetros del modelo: {'MLPC__hidden_layer_sizes': 13, 'MLPC__solve
r': 'lbfgs', 'MLPC__alpha': 1e-06, 'MLPC__activation': 'logistic'}
```

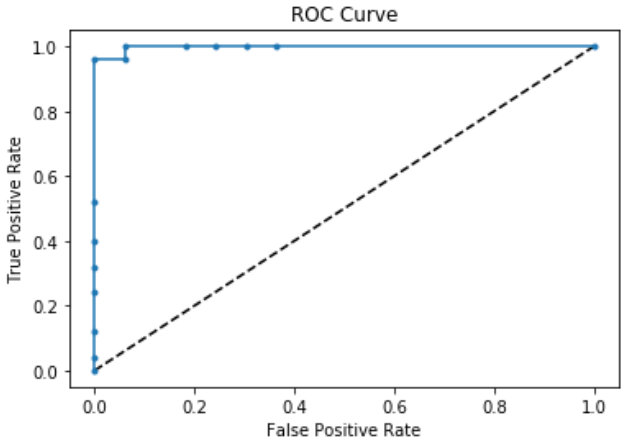
Métricas de Evaluación de Clasificación (Confusion Matrix, Accuracy, Precision, Recall, F1, ROC y AUC)

In [20]:
```
nn =  Classification_Metrics(MLPC_cv, X_test, y_test, y_test.unique())
```




```
Métricas del modelo:

Exactitud: 93.10%
Precisión: 86.21%
Sensibilidad: 100.00%
Puntaje F1: 92.59%

Alternativa con Classification_Report:
```

	precision	recall	f1-score	support
0	1.00	0.88	0.94	33
1	0.86	1.00	0.93	25
accuracy			0.93	58
macro avg	0.93	0.94	0.93	58
weighted avg	0.94	0.93	0.93	58




```
AUC: 99.76%
```

Support Vector Machine

Aplicamos el método de Máquina de Soporte Vectorial por medio del clasificador *SVC()* obtenido de los módulos de *sklearn*, construimos un *Pipeline*, primero con el comando de escalamiento de datos para posteriormente aplicar el clasificador, definimos nuestros parámetros del clasificador para realizar una búsqueda exhaustiva con validación cruzada, con un valor de *k*=10, para realizar la búsqueda utilizamos el comando *GridSearchCV()* proveniente de los módulos de sklearn para encontrar los mejores parámetros para nuestro modelo de clasificación.

Después de realizar la validación de parámetros debemos entrenar nuestro modelo con el conjunto de datos de entrenamiento.

In [21]:
```
# Pipeline
steps = [('scaler', StandardScaler()),
         ('SVM', SVC())]
pipeline = Pipeline(steps)
# Hyperparameters
parameters = {'SVM__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
              'SVM__C':[1, 10, 100],
              'SVM__gamma': [0.1, 0.01],
              'SVM__probability': [True]}
# Cross Validation
SVM_cv = GridSearchCV(pipeline, parameters, cv=10)
# Fit Model
SVM_cv.fit(X_train, y_train)

```

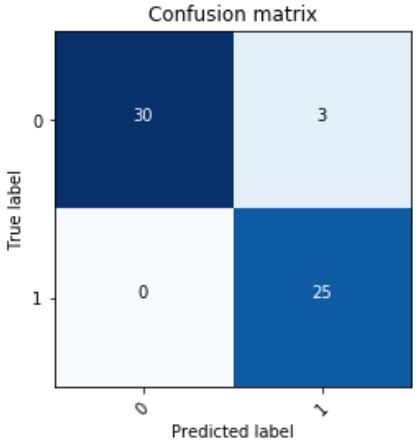
Out[21]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('SVM',
                                        SVC(C=1.0, break_ties=False,
                                            cache_size=200, class_weight=Non
e,
                                            coef0=0.0,
                                            decision_function_shape='ovr',
                                            degree=3, gamma='scale',
                                            kernel='rbf', max_iter=-1,
                                            probability=False,
                                            random_state=None, shrinking=Tru
e,
                                            tol=0.001, verbose=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'SVM__C': [1, 10, 100], 'SVM__gamma': [0.1, 0.01],
                         'SVM__kernel': ['linear', 'poly', 'rbf', 'sigmoid
'],
                         'SVM__probability': [True]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [22]:
```
print("Mejores Parámetros del modelo: {}".format(SVM_cv.best_params_))

```

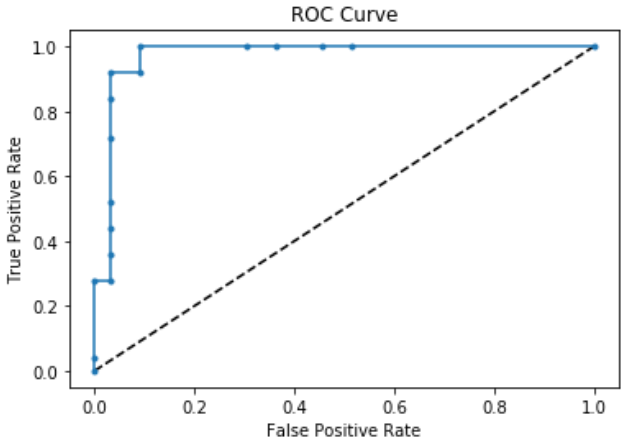
```
Mejores Parámetros del modelo: {'SVM__gamma': 0.1, 'SVM__probability': True,
'SVM__C': 1, 'SVM__kernel': 'sigmoid'}
```

Métricas de Evaluación de Clasificación (Confusion Matrix, Accuracy, Precision, Recall, F1, ROC y AUC)

In [23]:
```
svm =  Classification_Metrics(SVM_cv, X_test, y_test, y_test.unique())
```




```
Métricas del modelo:

Exactitud: 94.83%
Precisión: 89.29%
Sensibilidad: 100.00%
Puntaje F1: 94.34%

Alternativa con Classification_Report:

              precision    recall  f1-score   support

           0       1.00      0.91      0.95        33
           1       0.89      1.00      0.94        25

    accuracy                           0.95        58
   macro avg       0.95      0.95      0.95        58
weighted avg       0.95      0.95      0.95        58
```




```
AUC: 97.33%
```

Decision Tree Classifier

Aplicamos el método de Clasificación y Árboles de Regresiones por medio del clasificador *DecisionTreeClassifier()* obtenido de los módulos de *sklearn*, construimos un *Pipeline*, primero con el comando de escalamiento de datos para posteriormente aplicar el clasificador, definimos nuestros parámetros del clasificador para realizar una búsqueda exhaustiva con validación cruzada, con un valor de *k*=10, para realizar la búsqueda utilizamos el comando *GridSearchCV()* para encontrar los mejores parámetros para nuestro modelo de clasificación.

Después de realizar la validación de parámetros debemos entrenar nuestro modelo con el conjunto de datos de entrenamiento.

In [24]:
```
# Pipeline
steps = [('scaler', StandardScaler()),
         ('CART', DecisionTreeClassifier())] #
pipeline = Pipeline(steps)
# Hyperparameters
parameters = { "CART__criterion" : ['gini', 'entropy'],
               "CART__max_depth" : [4,6,8,12]}
# Cross Validation
CART_cv = GridSearchCV(pipeline, parameters, cv=10)
# Fit Model
CART_cv.fit(X_train, y_train)

```

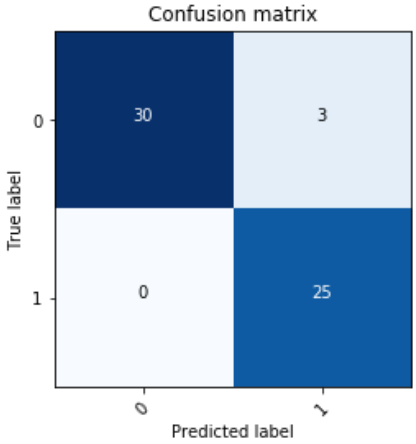
Out[24]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('scaler',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('CART',
                                        DecisionTreeClassifier(ccp_alpha=0.
0,
                                                               class_weight=
None,
                                                               criterion='gi
ni',
                                                               max_depth=Non
e,
                                                               max_features=
None,
                                                               max_leaf_node
s=None,
                                                               min_impurity_
decrease=0.0,
                                                               min_impurity_
split=None,
                                                               min_samples_l
eaf=1,
                                                               min_samples_s
plit=2,
                                                               min_weight_fr
action_leaf=0.0,
                                                               presort='depr
ecated',
                                                               random_state=
None,
                                                               splitter='bes
t'))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'CART__criterion': ['gini', 'entropy'],
                         'CART__max_depth': [4, 6, 8, 12]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [25]:
```
print("Mejores Parámetros del modelo: {}".format(CART_cv.best_params_))

```

```
Mejores Parámetros del modelo: {'CART__criterion': 'entropy', 'CART__max_dep
th': 4}
```

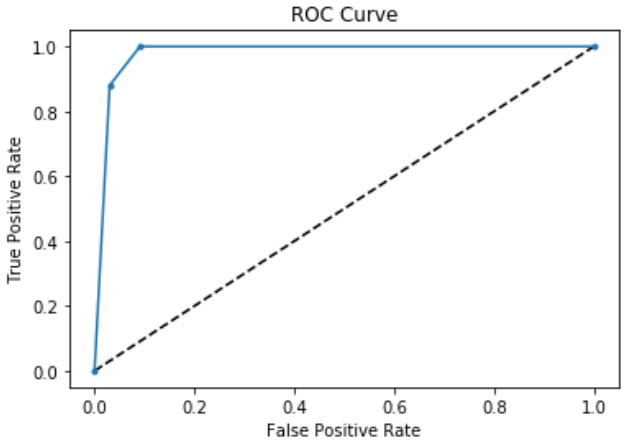
Métricas de Evaluación de Clasificación (Confusion Matrix, Accuracy, Precision, Recall, F1, ROC y AUC)

In [26]:
```
cart =  Classification_Metrics(CART_cv, X_test, y_test, y_test.unique())
```




```
Métricas del modelo:

Exactitud: 94.83%
Precisión: 89.29%
Sensibilidad: 100.00%
Puntaje F1: 94.34%

Alternativa con Classification_Report:
```

	precision	recall	f1-score	support
0	1.00	0.91	0.95	33
1	0.89	1.00	0.94	25
accuracy			0.95	58
macro avg	0.95	0.95	0.95	58
weighted avg	0.95	0.95	0.95	58




```
AUC: 97.94%
```

Naive Bayes

Aplicamos el método de Naive Bayes por medio del clasificador *GaussianNB()* obtenido de los módulos de *sklearn*, en este caso no es recomendable usar la validación cruzada de párametros ya que puede alenta el processo de computación y al ser un algoritmo de clasificación tan simple no es necesario.

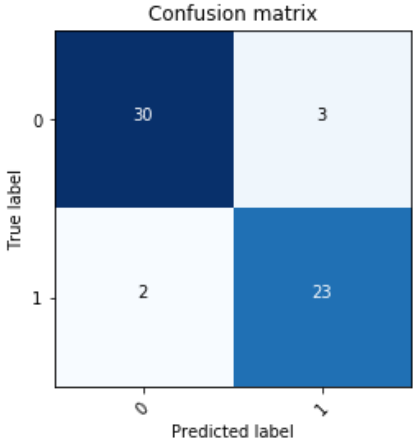
Construimos nuestro modelo de clasificación bayeseana con nuestro conjunto de entranamiento.

In [27]:
```
# Naive Bayes classifier
GNB = GaussianNB()
#Train the model using the training sets
GNB.fit(X_train, y_train)

```

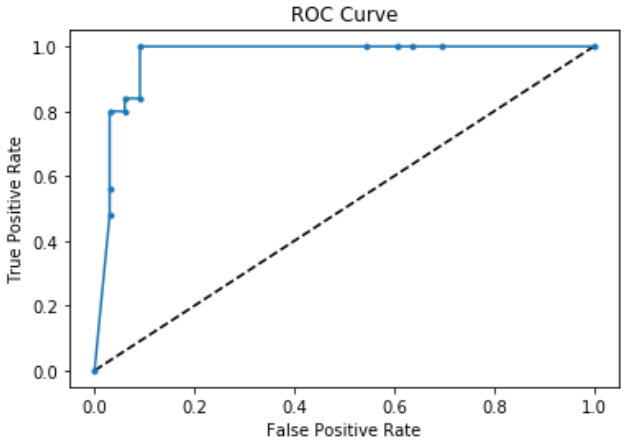
Out[27]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Métricas de Evaluación de Clasificación (Confusion Matrix, Accuracy, Precision, Recall, F1, ROC y AUC)

In [28]:
```
gnb =  Classification_Metrics(GNB, X_test, y_test, y_test.unique())

```




```
Métricas del modelo:

Exactitud: 91.38%
Precisión: 88.46%
Sensibilidad: 92.00%
Puntaje F1: 90.20%

Alternativa con Classification_Report:

              precision    recall  f1-score   support

           0       0.94      0.91      0.92        33
           1       0.88      0.92      0.90        25

    accuracy                           0.91        58
   macro avg       0.91      0.91      0.91        58
weighted avg       0.91      0.91      0.91        58
```




```
AUC: 96.61%
```

Conclusión

In [30]:
```
d = {
     "Classifier" : ["K-NN", "Log-Reg", "Neu-Net", "SVM", "CART", "G-NB"],
     "Accuracy": [knn["accuracy"], logreg["accuracy"], nn["accuracy"],
                  svm["accuracy"], cart["accuracy"], gnb["accuracy"]],
     "Precision": [knn["precision"], logreg["precision"], nn["precision"],
                   svm["precision"], cart["precision"], gnb["precision"]],
     "Recall": [knn["recall"], logreg["recall"], nn["recall"], svm["recall"],
                cart["recall"], gnb["recall"]],
     "F1": [knn["f1"], logreg["f1"], nn["f1"], svm["f1"], cart["f1"], gnb["f1"],
     "AUC": [knn["AUC"], logreg["AUC"], nn["AUC"], svm["AUC"], cart["f1"],
    }

df_scores = pd.DataFrame(d)
df_scores = df_scores.set_index(["Classifier"])
df_scores = df_scores.sort_values(by=['F1'], ascending=False)
df_scores

```

Out[30]:

Classifier	AUC	Accuracy	F1	Precision	Recall
SVM	0.973333	0.948276	0.943396	0.892857	1.00
CART	0.943396	0.948276	0.943396	0.892857	1.00
Neu-Net	0.997576	0.931034	0.925926	0.862069	1.00
G-NB	0.901961	0.913793	0.901961	0.884615	0.92
K-NN	0.936364	0.879310	0.872727	0.800000	0.96
Log-Reg	0.986667	0.879310	0.872727	0.800000	0.96

Utilizando las medidas de calidad podemos obtener una comparación de los clasificadores, obtenemos que el clasificador con mayor puntaje *F1* es el método de *Clasificador de Árboles de Regresión*, el segundo mejor fue el método de *Redes Neuronales*, seguido de *Máquina de Soporte Vectorial*,*Clasificador Bayesiano*, *K - Vecinos más cercanos* y por último *Regresión Logística*. El mejor clasificador fue *CART* porque es un método que permite cortar ramas de ábol por medio de algoritmos voraces tiene una mayor precisión en la predicción de una clase binaria. Además de obtener mejores resultados en todas sus medidas de calidad en comparación a los otros clasificadores.

El proyecto funciona para aplicar todo lo visto previamente durante todo el curso y con el apoyo de los cursos de *DataCamp*, además de ser esta materia una continuidad directa de *Aprendizaje estadístico*, los temas quedaron mejor entendidos y se aplicaron todas las técnicas aprendidas a lo largo de los cursos, además de aplicar lo visto en el primer parcial para procesar e interpretar de mejor manera los datos para poder construir correctamente matrices y vectores de características para poder clasificarlos en clases de acuerdo a modelos de aprendizaje supervidado con una mayor precisión y mejor estructura de trabajo.

Referencias Bibliográficas

- 1. Source: Congressional Quarterly Almanac, 98th Congress, 2nd session 1984, Volume XL: Congressional Quarterly Inc. Washington, D.C., 1985.
  2. Donor: Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)
  3. Date: 27 April 1987
  4. Url: Congressional Voting Records Data Set (https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records)
- Onel Harrison. (2018). Machine Learning Basics with the K-Nearest Neighbors Algorithm. (Mayo 23, 2020), de Towards Data Science Sitio web: URL (https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761)
- Saishruthi Swaminathan. (2018). Logistic Regression — Detailed Overview. (Mayo 23, 2020), de Towards Data Science Sitio web: URL (https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc)
- Tony Yiu. (2019). Understanding Neural Networks. (Mayo 23, 2020), de Towards Data Science Sitio web: URL (https://towardsdatascience.com/understanding-neural-networks-19020b758230)
- Ajay Yadav. (2018). SUPPORT VECTOR MACHINES(SVM). (Mayo 23, 2020), de Towards Data Science Sitio web: URL (https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589)
- Desconocido. (Desconocido). Arboles de Decisión - Parte I. (Mayor 23, 2020), de bookdown Sitio web: URL (https://bookdown.org/content/2031/arboles-de-decision-parte-i.html)
- Rohith Gandhi. (2018). Naive Bayes Classifier. (Mayo 23, 2020), de Towards Data Science Sitio web: URL (https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c)
- Desconocido. (Desconocido). Machine Learning: Seleccion Metricas de clasificacion. (Mayo 23, 2020), de Sitio Big Data Sitio web: URL (https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3/)