# Pràctica 2:
# WATER QUALITY

Rubén Simó Marín – 1569391
Sergio Navarrete Villalta – 1564572
Alejandro García García – 1564537

# 1. 1. EDA (exploratory data analysis)

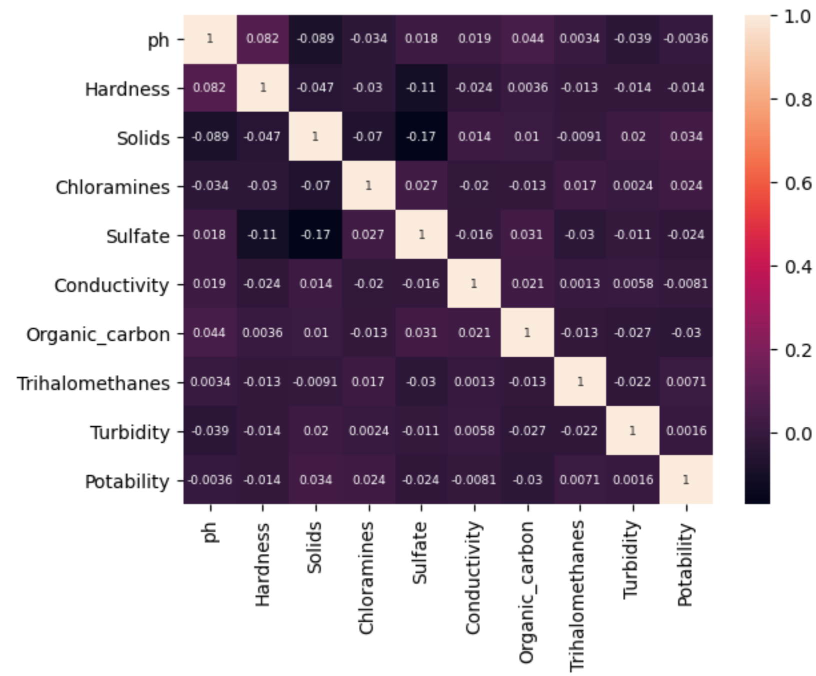
```
dataset
```

| | ph | Hardness | Solids | Chloramines | Sulfate | Conductivity | Organic_carbon | Trihalomethanes | Turbidity | Potability |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | 204.890 | 20791.319 | 7.300 | 368.516 | 564.309 | 10.380 | 86.991 | 2.963 | 0 |
| **1** | 3.716 | 129.423 | 18630.058 | 6.635 | NaN | 592.885 | 15.180 | 56.329 | 4.501 | 0 |
| **2** | 8.099 | 224.236 | 19909.542 | 9.276 | NaN | 418.606 | 16.869 | 66.420 | 3.056 | 0 |
| **3** | 8.317 | 214.373 | 22018.417 | 8.059 | 356.886 | 363.267 | 18.437 | 100.342 | 4.629 | 0 |
| **4** | 9.092 | 181.102 | 17978.986 | 6.547 | 310.136 | 398.411 | 11.558 | 31.998 | 4.075 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **3271** | 4.668 | 193.682 | 47580.992 | 7.167 | 359.949 | 526.424 | 13.894 | 66.688 | 4.436 | 1 |
| **3272** | 7.809 | 193.553 | 17329.802 | 8.061 | NaN | 392.450 | 19.903 | NaN | 2.798 | 1 |
| **3273** | 9.420 | 175.763 | 33155.578 | 7.350 | NaN | 432.045 | 11.039 | 69.845 | 3.299 | 1 |
| **3274** | 5.127 | 230.604 | 11983.869 | 6.303 | NaN | 402.883 | 11.169 | 77.488 | 4.709 | 1 |
| **3275** | 7.875 | 195.102 | 17404.177 | 7.509 | NaN | 327.460 | 16.140 | 78.698 | 2.309 | 1 |

```python
print("Tipus d'atributs:")
dataset.dtypes
```

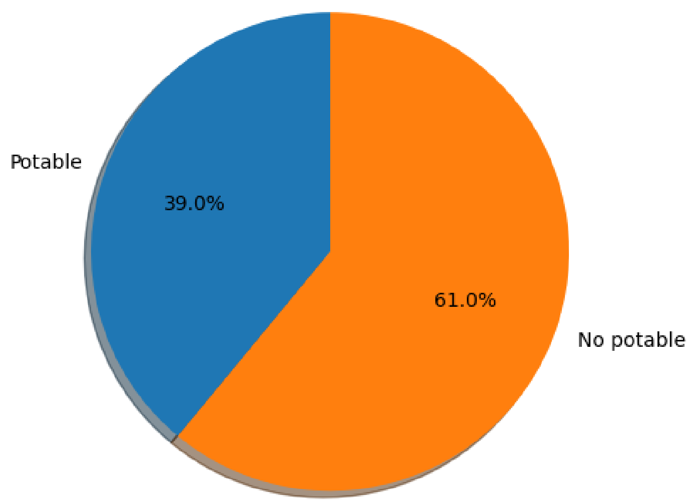
```
Tipus d'atributs:
ph                 float64
Hardness           float64
Solids             float64
Chloramines        float64
Sulfate            float64
Conductivity       float64
Organic_carbon     float64
Trihalomethanes    float64
Turbidity          float64
Potability           int64
dtype: object
```

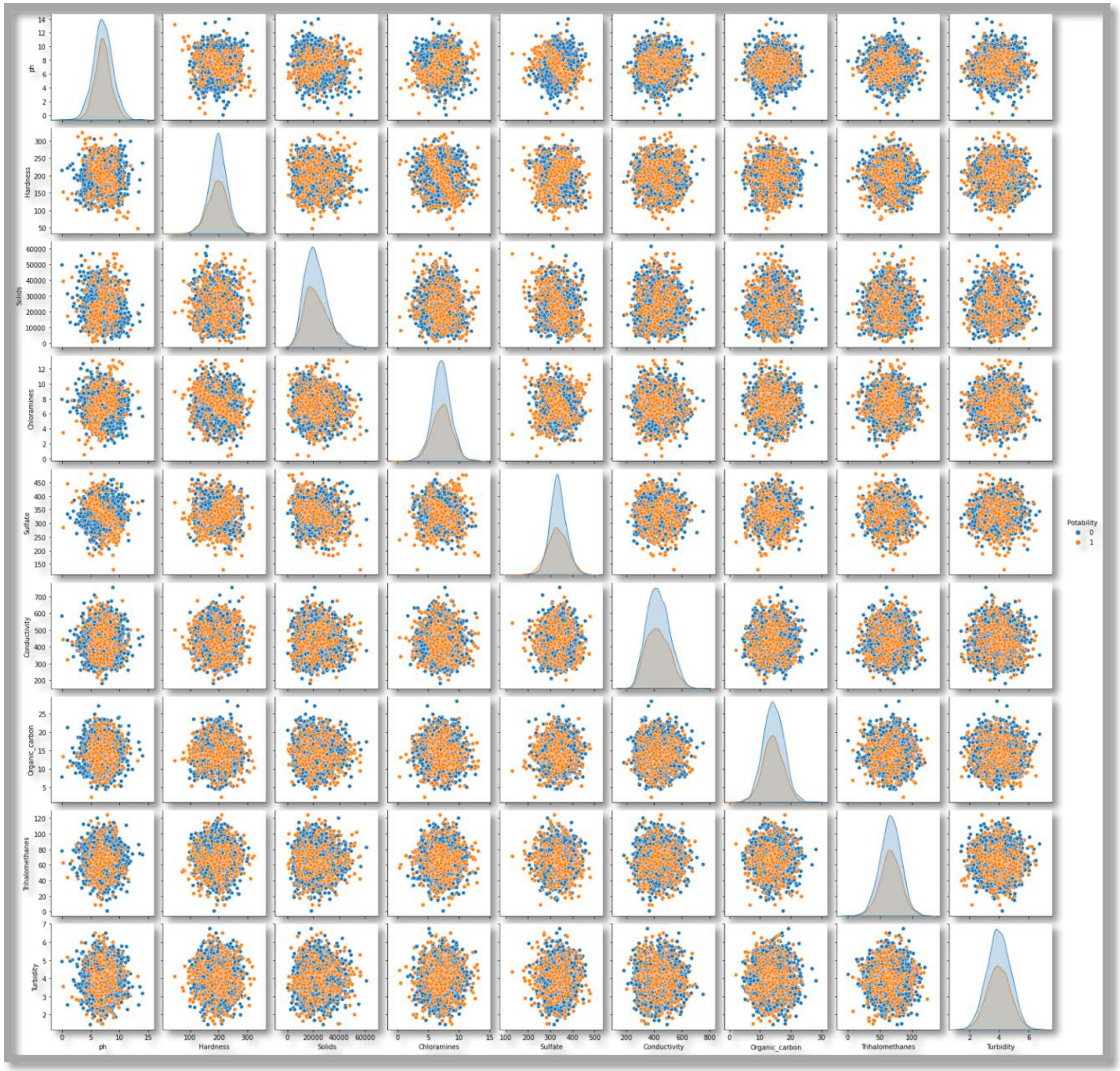
```python
correlacio = dataset.corr()
plt.figure()
ax = sns.heatmap(correlacio, annot=True, annot_kws={"size": 6.5})
```



```
0    1998
1    1278
Name: Potability, dtype: int64
```

```
potable = dataset[dataset.Potability == 1]
noPotable = dataset[dataset.Potability == 0]

fig,ax1 = plt.subplots()
labels = 'Potable', 'No potable'
ax1.pie([potable.shape[0],noPotable.shape[0]], labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal')

plt.show()
```




ph
Hardness
Solids
Chloramines
Sulfate
Conductivity
Organic_carbon
Trihalomethanes
Turbidity
Potability
0
1

# 1. 2. Preprocessing (normalitzation, outlier removal, feature selection..)

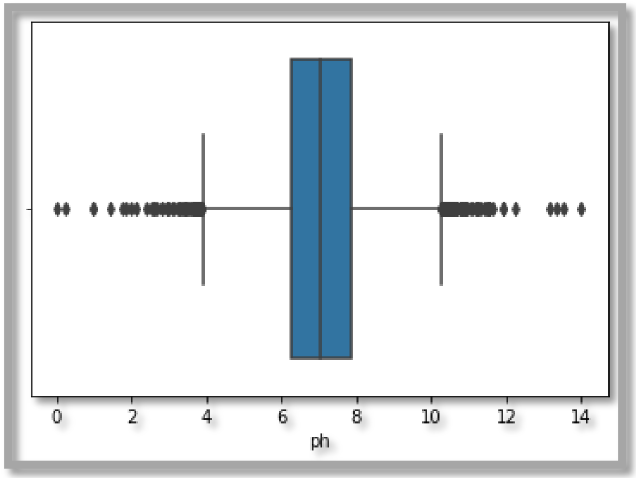
```
dataset.isnull().sum()
```

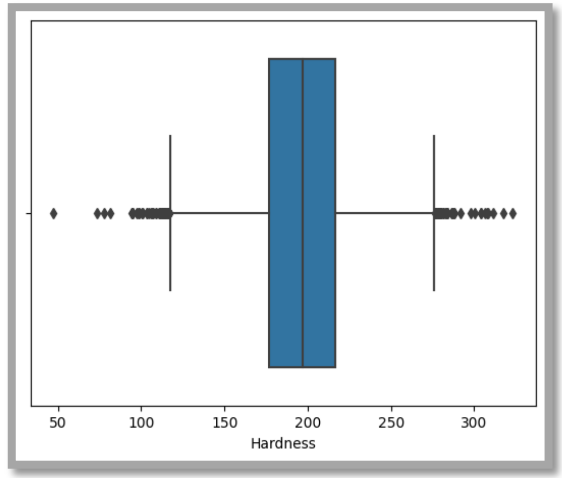
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

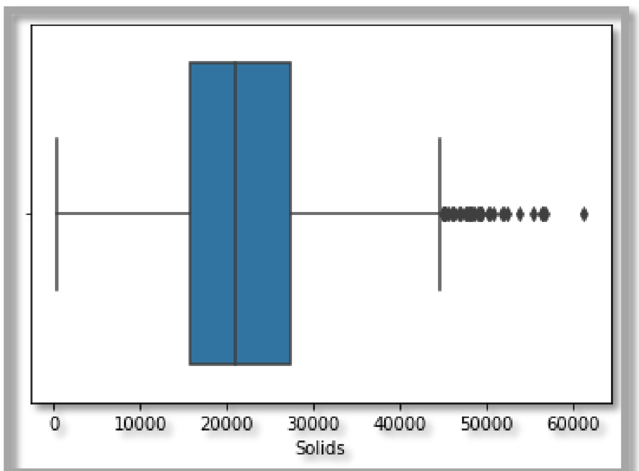
```
dataset['ph'].fillna(value=dataset['ph'].median(),inplace=True)
dataset['Sulfate'].fillna(value=dataset['Sulfate'].median(),inplace=True)
dataset['Trihalomethanes'].fillna(value=dataset['Trihalomethanes'].median(),inplace=True)
```

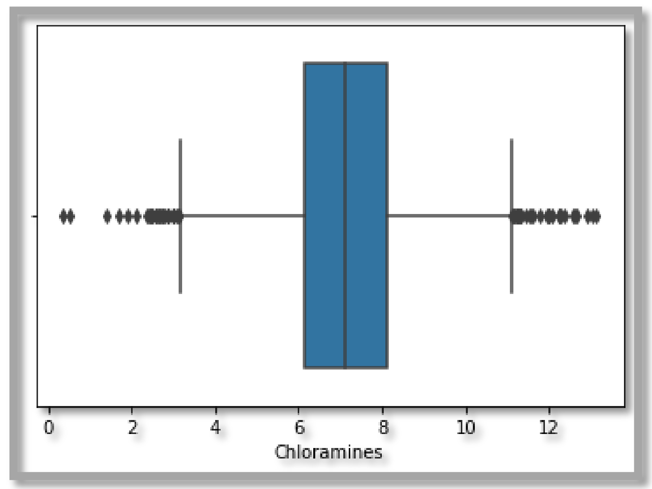
```
dataset.isnull().sum()
```

```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```


0 2 4 6 8 10 12 14
ph


50 100 150 200 250 300
Hardness


0 10000 20000 30000 40000 50000 60000
Solids


0 2 4 6 8 10 12
Chloramines

```python
#normalitzacio de dades utilitzant preprocessing

from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

scaler = StandardScaler()
standardized = scaler.fit_transform(dataset)
standardized_dataset = pd.DataFrame(standardized, columns=dataset.columns)

print(standardized_dataset)
```

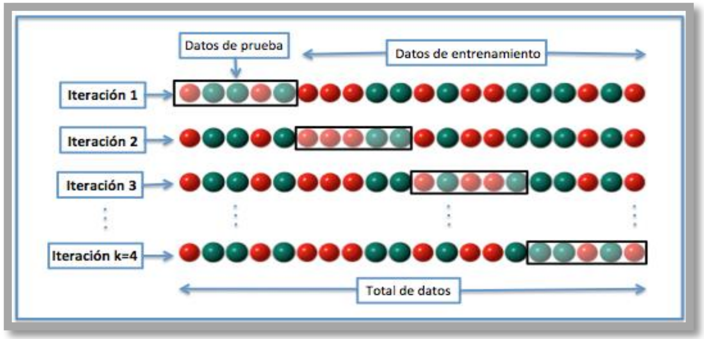
```
          ph  Hardness  Solids  Chloramines  Sulfate  Conductivity  \
0     -0.025     0.259  -0.139        0.112    0.966         1.709
1     -2.285    -2.036  -0.386       -0.308   -0.015         2.063
2      0.697     0.848  -0.240        1.361   -0.015        -0.094
3      0.845     0.548   0.000        0.592    0.644        -0.779
4      1.373    -0.464  -0.460       -0.364   -0.650        -0.344
...      ...       ...     ...          ...      ...           ...
3271 -1.637    -0.082   2.916        0.028    0.729         1.240
3272  0.500    -0.086  -0.534        0.593   -0.015        -0.418
3273  1.596    -0.627   1.271        0.144   -0.015         0.072
3274 -1.325     1.041  -1.144       -0.517   -0.015        -0.289
3275  0.545    -0.039  -0.526        0.245   -0.015        -1.222

      Organic_carbon  Trihalomethanes  Turbidity  Potability
0             -1.181            1.305     -1.286      -0.800
1              0.271           -0.639      0.684      -0.800
2              0.781            0.001     -1.167      -0.800
3              1.255            2.152      0.848      -0.800
4             -0.824           -2.182      0.139      -0.800
...              ...              ...        ...         ...
3271          -0.118            0.018      0.601       1.250
3272           1.699            0.014     -1.498       1.250
3273          -0.981            0.218     -0.856       1.250
3274          -0.942            0.703      0.951       1.250
3275           0.561            0.780     -2.124       1.250

[3276 rows x 10 columns]
```

# 1. 3. Model Selection

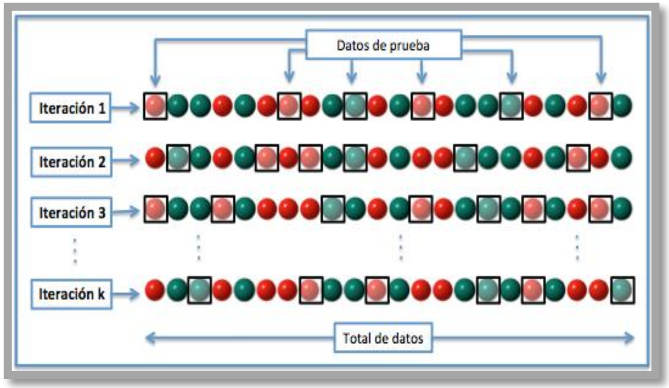
- – Logistic Regression
- – Nearest K Neighbors
- – SVM

# 1. 4. Cross-validation

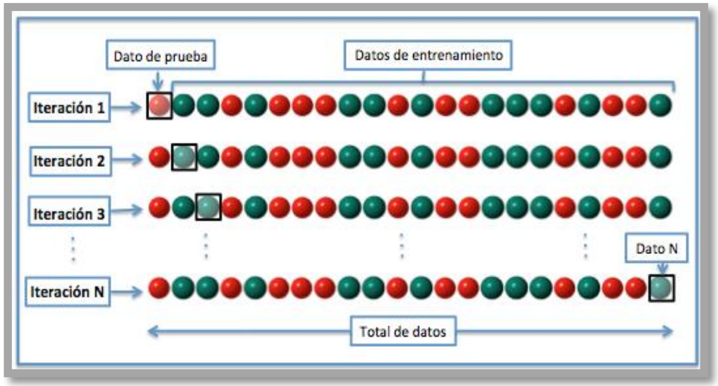
- \- Cross-validation k-fold




- \- Cross-validation sufflesplit




- \- Leave-one-out



```python
folds = range(2,31)

def evaluatemodel(cv, standar):

    x = standar[:,[0,1,2,3,4,5,6,7,8]]
    y = standar[:,9]
    lab = preprocessing.LabelEncoder()
    y_transformed = lab.fit_transform(y)

    logReg = LogisticRegression()
    scores = cross_val_score(logReg, x, y_transformed, scoring='accuracy', cv=cv, n_jobs=-1)
    return mean(scores), scores.min(), scores.max()

for k in folds:
    cv = KFold(n_splits=k, shuffle=True, random_state=10)
    k_mean, k_min, k_max = evaluatemodel(cv, standardized)
    print('-> folds=%d, accuracy=%.3f (%.3f,%.3f)' % (k, k_mean, k_min, k_max))
```

```
-> folds=2, accuracy=0.611 (0.596,0.626)
-> folds=3, accuracy=0.614 (0.589,0.631)
-> folds=4, accuracy=0.611 (0.584,0.628)
-> folds=5, accuracy=0.611 (0.569,0.631)
-> folds=6, accuracy=0.611 (0.577,0.632)
-> folds=7, accuracy=0.610 (0.577,0.643)
```

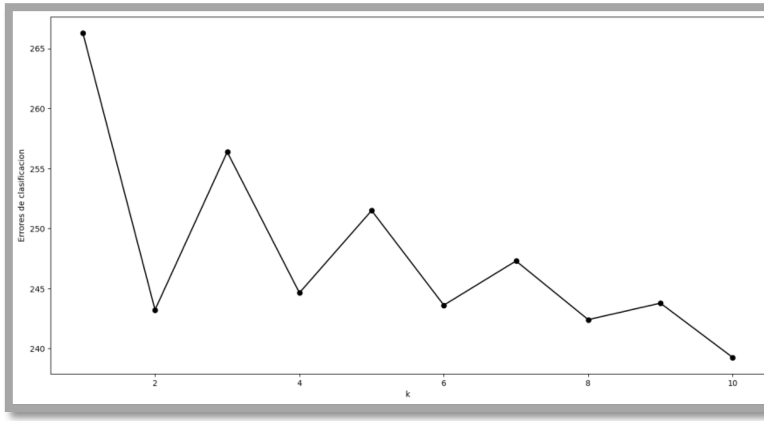
# 1. 5. Metric Analysis

- -Accuracy_score
- -F1_score
- -Average_precision_score

```python
def PRCurve(y_v,probs, n_classes):
    precision = {}
    recall = {}
    average_precision = {}
    plt.figure()
    for i in range(n_classes):
        precision[i], recall[i], _ = precision_recall_curve(y_v == i, probs[:, i])
        average_precision[i] = average_precision_score(y_v == i, probs[:, i])

        plt.plot(recall[i], precision[i],
        label='Precision-recall curve of class {0} (area = {1:0.2f})'
                               ''.format(i, average_precision[i]))
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.legend(loc="upper right")

def RocCurve(y_v, probs, n_classes):
    fpr = {}
    tpr = {}
    roc_auc = {}
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_v == i, probs[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    # Plot ROC curve
    plt.figure()
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})' ''.format(i, roc_auc[i]))
    plt.legend()
```

```
LOGISTIC REGRESION

Logistic Regression Score:  0.6036585365853658
Logistic Regression Cross Val Score:  0.611450559762622

260 Errores de clasificacion de un total de 656
396 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[396   0]
 [260   0]]

Logistic Regression F1 Score: 0.6036585365853658
```

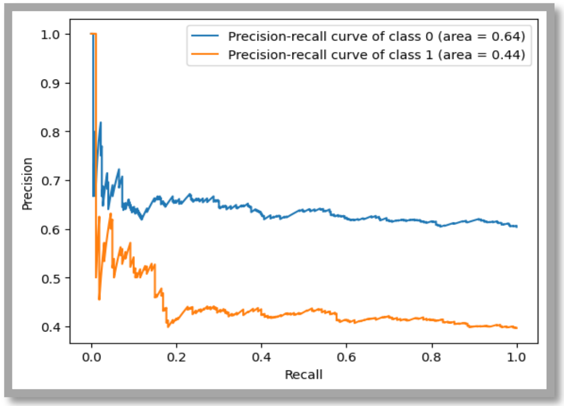
```
NEAREST K NEIGHBORS

k=1: 266.301 errores de clasificación de un total de 655
k=2: 243.225 errores de clasificación de un total de 655
k=3: 256.373 errores de clasificación de un total de 655
k=4: 244.64 errores de clasificación de un total de 655
k=5: 251.497 errores de clasificación de un total de 655
k=6: 243.613 errores de clasificación de un total de 655
k=7: 247.296 errores de clasificación de un total de 655
k=8: 242.405 errores de clasificación de un total de 655
k=9: 243.781 errores de clasificación de un total de 655
k=10: 239.26 errores de clasificación de un total de 655
```

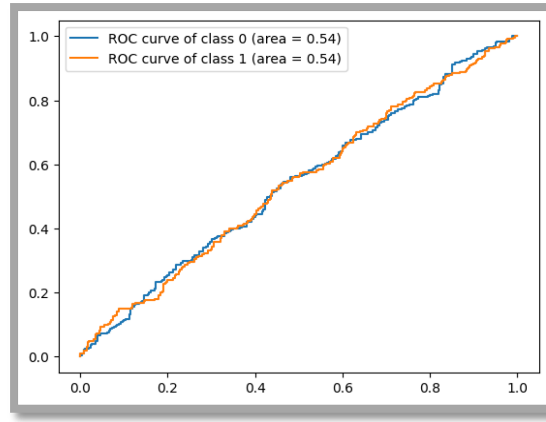


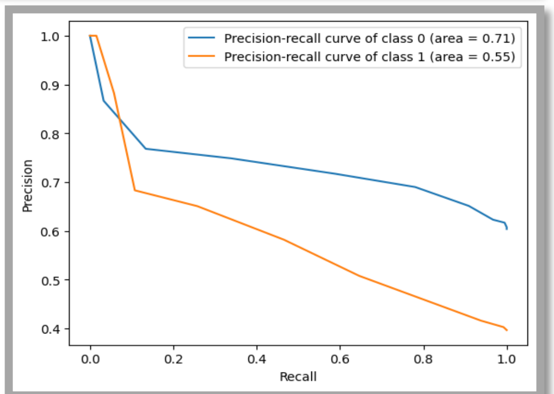

```
Nearest K Neighbour Score:  0.6509146341463414
Nearest K Neighbour Cross Val Score:  0.6389266782175319

229 Errores de clasificacion de un total de 656
427 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[360  36]
 [193  67]]

Nearest K Neighbour F1 Score: 0.6509146341463414
```

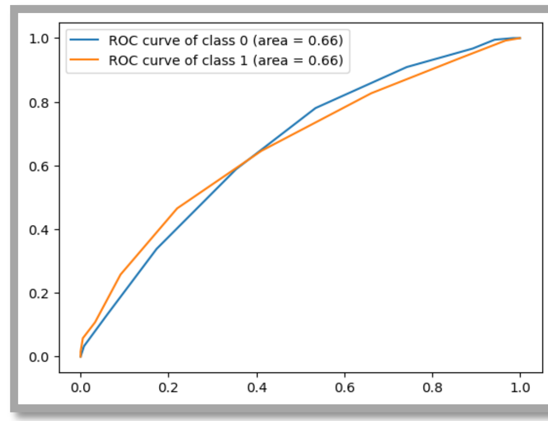
```
SVM with rbf kernel Score:  0.6935975609756098
SVM with rbf kernel Cross Val Score:  0.6595395555974951

201 Errores de clasificacion de un total de 656
455 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[381  15]
 [186  74]]

SVM with rbf kernel F1 Score: 0.6935975609756098
```

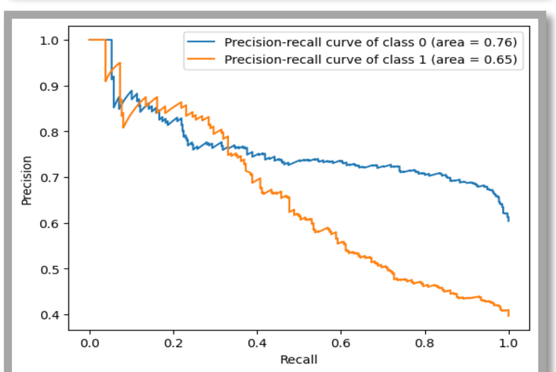
```
SVM Linear Score:  0.6036585365853658
SVM Linear Cross Val Score:  0.611450559762622

260 Errores de clasificacion de un total de 656
396 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[396   0]
 [260   0]]

SVM Linear F1 Score: 0.6036585365853658
```

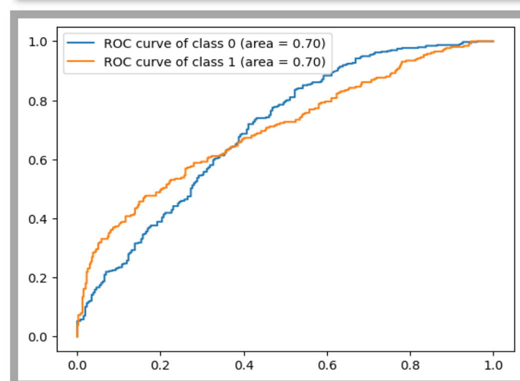

Precision-recall curve of class 0 (area = 0.64)
Precision-recall curve of class 1 (area = 0.44)
Precision
Recall


ROC curve of class 0 (area = 0.54)
ROC curve of class 1 (area = 0.54)


Precision-recall curve of class 0 (area = 0.71)
Precision-recall curve of class 1 (area = 0.55)
Precision
Recall


ROC curve of class 0 (area = 0.66)
ROC curve of class 1 (area = 0.66)


Precision-recall curve of class 0 (area = 0.76)
Precision-recall curve of class 1 (area = 0.65)
Precision
Recall


ROC curve of class 0 (area = 0.70)
ROC curve of class 1 (area = 0.70)

- Logistic Regression

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| class 0 | 0.60 | 1.00 | 0.75 | 396 |
| class 1 | 0.00 | 0.00 | 0.00 | 260 |
| accuracy | | | 0.60 | 656 |
| macro avg | 0.30 | 0.50 | 0.38 | 656 |
| weighted avg | 0.36 | 0.60 | 0.45 | 656 |

- Nearest K Neighbors

Nearest K Neighbour

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| class 0 | 0.65 | 0.91 | 0.76 | 396 |
| class 1 | 0.65 | 0.26 | 0.37 | 260 |
| accuracy | | | 0.65 | 656 |
| macro avg | 0.65 | 0.58 | 0.56 | 656 |
| weighted avg | 0.65 | 0.65 | 0.60 | 656 |

- SVM (SVM amb rbf)

SVM

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| class 0 | 0.67 | 0.96 | 0.79 | 396 |
| class 1 | 0.83 | 0.28 | 0.42 | 260 |
| accuracy | | | 0.69 | 656 |
| macro avg | 0.75 | 0.62 | 0.61 | 656 |
| weighted avg | 0.74 | 0.69 | 0.65 | 656 |

# 1. 6. Hyperparameter Search

- Exhaustive Grid Search

- Randomized Parameter Optimization

- Bayesian optimization search

```python
# GridSearchCV
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':('linear', 'rbf'), 'C':[0,0.2,0.5,0.75,1,1.25,1.5], 'gamma':[0.09,0.08,0.07,0.1]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
search = clf.fit(x_train,y_train)
print("GridSearchCV: ", search.best_params_)

# RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV

clf = RandomizedSearchCV(svc, parameters, random_state=0)
search = clf.fit(x_train,y_train)
print("RandomizedSearchCV: ", search.best_params_)
```

```
GridSearchCV:  {'C': 1.25, 'gamma': 0.1, 'kernel': 'rbf'}
RandomizedSearchCV:  {'kernel': 'rbf', 'gamma': 0.09, 'C': 1}
```

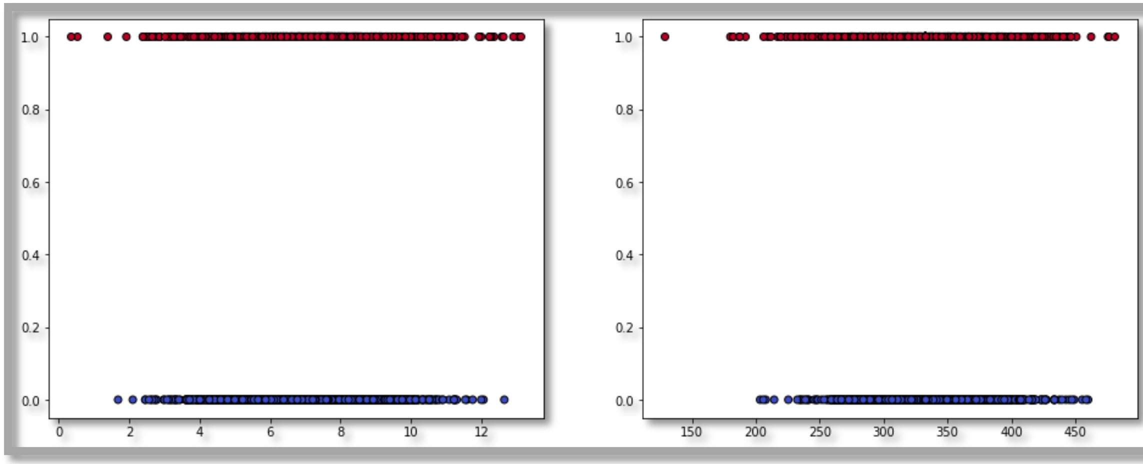
## GridSearchCV

```
SVM with rbf kernel Score:  0.6966463414634146
SVM with rbf kernel Cross Val Score:  0.6599222544632911

199 Errores de clasificacion de un total de 656
457 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[376  20]
 [179  81]]

SVM with rbf kernel F1 Score: 0.6966463414634146
```

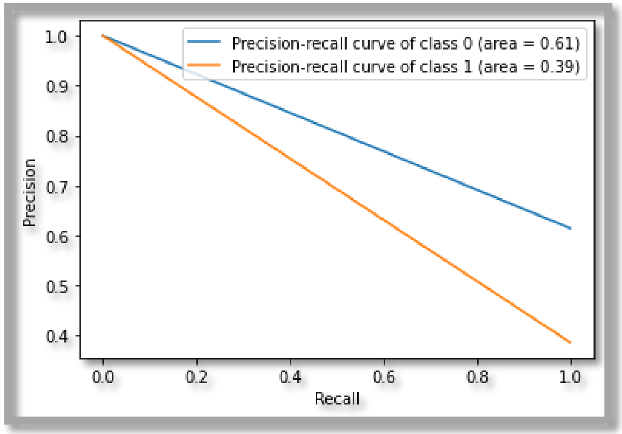
## RandomizedSearchCV

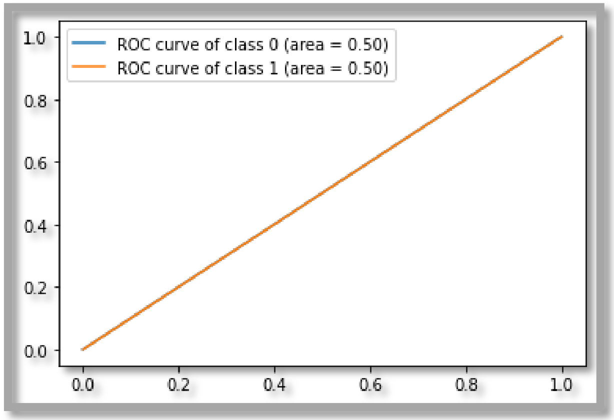
```
SVM with rbf kernel Score:  0.6905487804878049
SVM with rbf kernel Cross Val Score:  0.6591594779568074

203 Errores de clasificacion de un total de 656
453 Aciertos de clasificacion de un total de 656

Confusion matrix:
[[379  17]
 [186  74]]

SVM with rbf kernel F1 Score: 0.6905487804878049
```

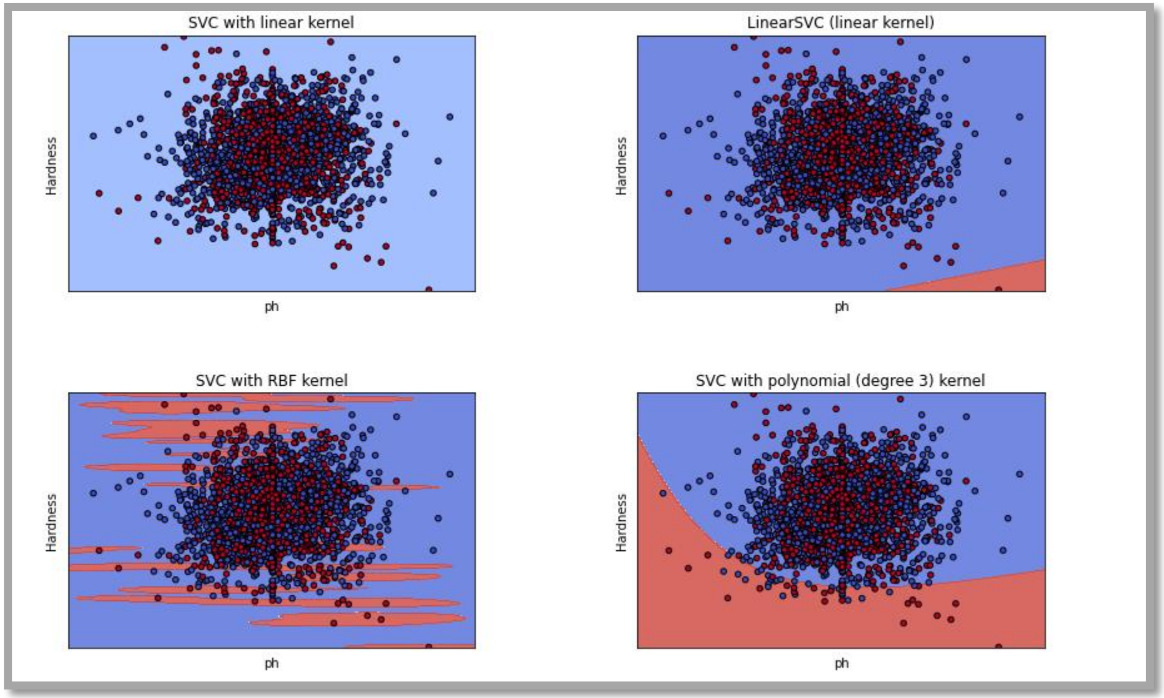
# 2. Apartat (A): Comparativa de models




```
Correct classification Logistic  0.5 % of the data:  0.6153846153846154
Correct classification SVM       0.5 % of the data:  0.5634920634920635
Correct classification Logistic  0.7 % of the data:  0.5940996948118006
Correct classification SVM       0.7 % of the data:  0.5534079348931842
Correct classification Logistic  0.8 % of the data:  0.6448170731707317
Correct classification SVM       0.8 % of the data:  0.573170731707317
```


Precision-recall curve of class 0 (area = 0.61)
Precision-recall curve of class 1 (area = 0.39)
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve of class 0 (area = 0.50)
ROC curve of class 1 (area = 0.50)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


SVC with linear kernel
Hardness
ph
LinearSVC (linear kernel)
Hardness
ph
SVC with RBF kernel
Hardness
ph
SVC with polynomial (degree 3) kernel
Hardness
ph