**Key Performance Indices (KPI’s) and Mushroom Picking**

In this practice, you will compute the **Confusion Matrix** and other **KPI**'s.  
You will demonstrate how the **Threshold** affects the **F1** score, plot the **ROC** (Receiver Operating Characteristic) curve and calculate the **AUC** (Area Under the Curve) score.

**Downloads, Imports, and Definitions**

Try to use Colab for this practice. First update the needed packages: Plotly and Pandas.

One problem when exporting Jupiter notebooks to HTML format, is that Plotly graphs disappear.  
To solve it, the graphs need to be converted to images, using orca. – its installation instructions are at [Install orca on Google Colab](https://plotly.com/python/orca-management/#install-orca-on-google-colab).

Now, install our regular packages: **numpy, pandas, seaborn, plotly.express, matplotlib.pyplot, and sklearn imports:**

Metrics, pipeline, linear\_model, preprocessing, and neural\_network.

make\_pipeline,SGDClassifier, OneHotEncoder, OrdinalEncoder, MLPClassifier, LogisticRegression, train\_test\_split

In [3]:

**Data Exploration**

For this practical you will use the [*Mushroom* dataset](https://archive.ics.uci.edu/ml/datasets/Mushroom) from 1987.

**Dataset Information**

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms from the Agaricus and Lepiota Family.  
Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended.  
This latter class (‘unknown edibility / not recommended’) was combined with the poisonous one.  
The Guide clearly states that there is no simple rule for determining the edibility of a mushroom. So in real world don’t try rules like ``leaflets three, let it be'' for Poisonous Oak and Ivy.



**Attribute Information**

1. **cap-shape**: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2. **cap-surface**: fibrous=f, grooves=g, scaly=y, smooth=s
3. **cap-color**: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u,red=e,white=w,yellow=y
4. **bruises**?: bruises=t, no=f
5. **odor**: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
6. **gill-attachment**: attached=a, descending=d, free=f, notched=n
7. **gill-spacing**: close=c, crowded=w, distant=d
8. **gill-size**: broad=b, narrow=n
9. **gill-color**: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10. **stalk-shape**: enlarging=e, tapering=t
11. **stalk-root**: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
12. **stalk-surface-above-ring**: fibrous=f, scaly=y, silky=k, smooth=s
13. **stalk-surface-below-ring**: fibrous=f, scaly=y, silky=k, smooth=s
14. **stalk-color-above-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15. **stalk-color-below-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. **veil-type**: partial=p, universal=u
17. **veil-color**: brown=n, orange=o, white=w, yellow=y
18. **ring-number**: none=n, one=o, two=t
19. **ring-type**: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20. **spore-print-color**: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21. **population**: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22. **habitat**: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

**Traget Information**

* **type**: poisonous=p, edible=e



Download the dataset from Github and explore it with Pandas tools, and print its df info

**Data Pre-processing**

The dataset includes only categorical data, so feature encoding is needed.  
Prepare several encodings now to be used later: OneHotEncoder, DummyEncoder and OrdinalEncoder

print the categories of the encoding.

An option: flatten the N-D array into 1-D array that can be used as column names for the pre-processed DataFrame.

Print the DataFrame after encoding

As in previous praticals , separate the target from the original DataFrame.

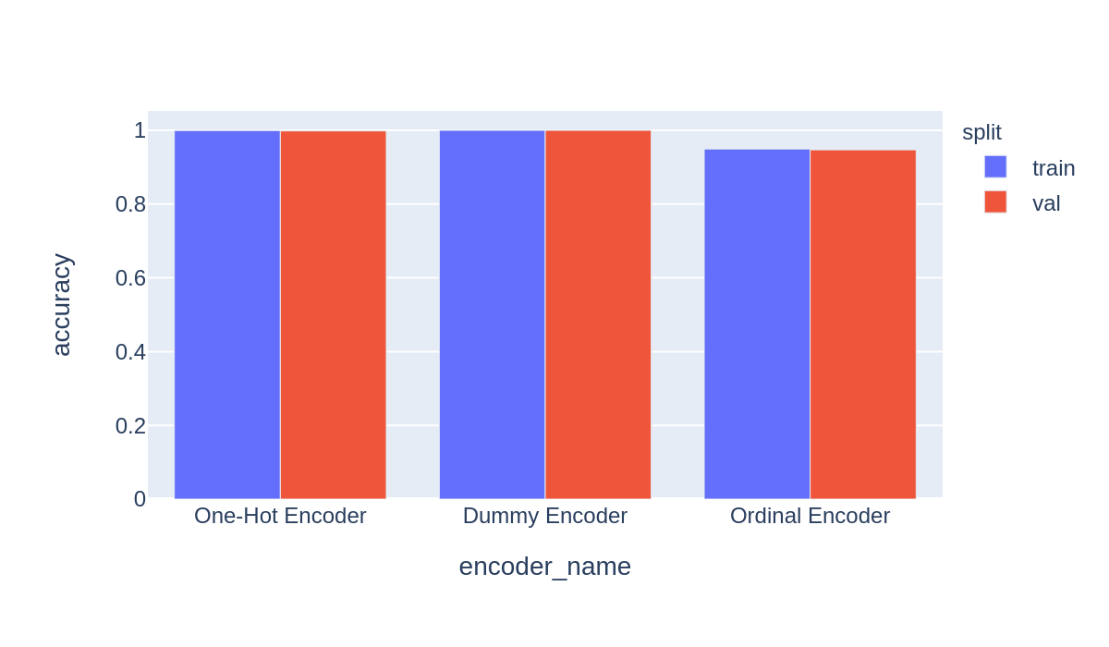
And split the data to train and validation.

**Model Training**

Train SGDClassifier and check which encoding is better on this dataset.

Try as practice converting the graphs of Plotly (showing accuracy vs. encoder) into images using [Get Image as Bytes](https://plotly.com/python/static-image-export/#get-image-as-bytes) and then printing the image with [IPython.display.Image](https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html#IPython.display.Image). Remember to show accuracy for both training and validation sets.

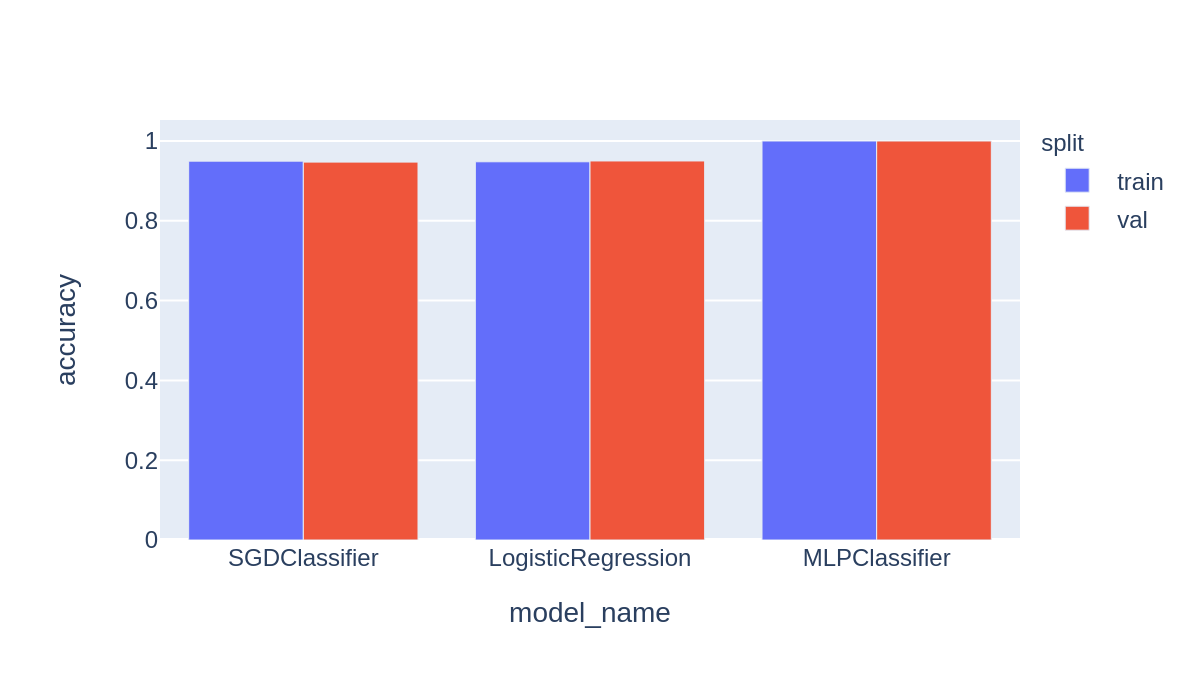
An example for such graph is:



Now compare models, using the same encoding for all: the OrdinalEncoding,  to see which model is better for this dataset.

Compare SGDClassifier, LogisticRegression and MLPClassifier.

An example for such graph is:



**Reporting performance: the KPI’s**

KPI’s are used to **explain** the results of a classifier.

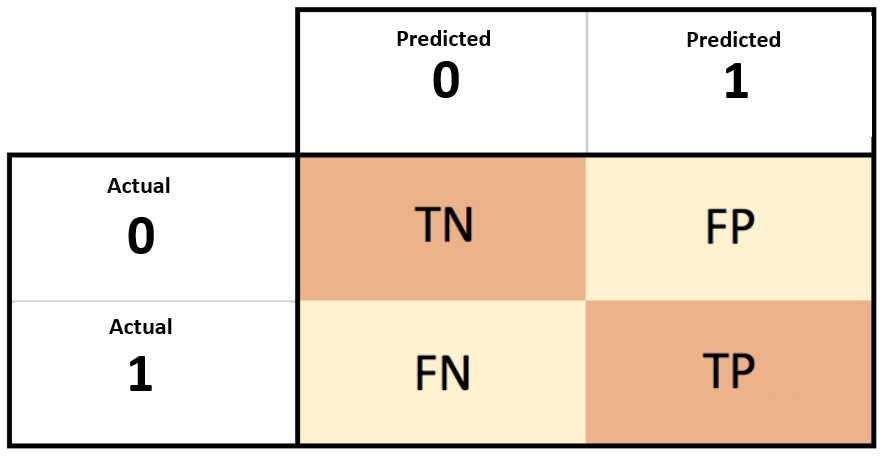
**Confusion Matrix**

Starting with confusion matrix,

For a binary case (two classes), the classification can yield one out of four outcomes:

1. **TP (True Positive**) - The model classified correctly that a sample is positive.
2. **TN (True Negative)** - The model classified correctly that a sample is negative.
3. **FP (False Positive)** - The model classified a sample as positive but the sample is actually negative.
4. **FN (False Negative)** - The model classified a sample as negative but the sample is actually positive.

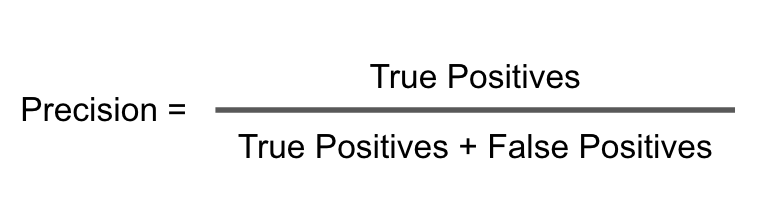
We can show these values in a matrix:



This matrix is called [**Confusion Matrix**](https://en.wikipedia.org/wiki/Confusion_matrix).  
With these values we can calculate a few **KPI**'s (Key Performance Indicators):

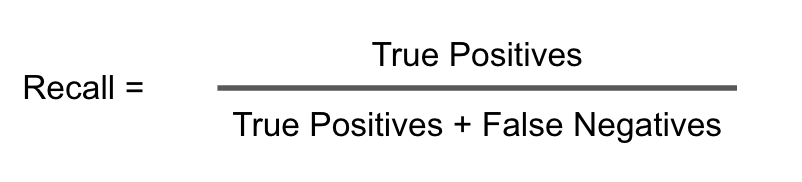
**Precision**

The percentage of correct positive predictions out of all the positive predictions.



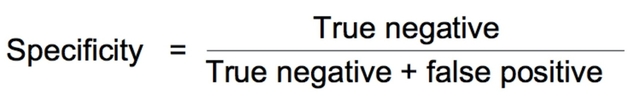
**Recall**

The percentage of correct positive predictions out of all the actual positive samples.



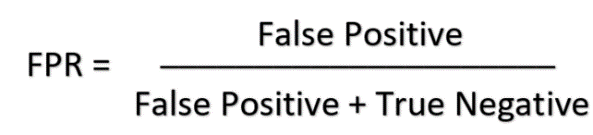
**Specificity**

The percentage of correct negative predictions out of all the actual negative samples.



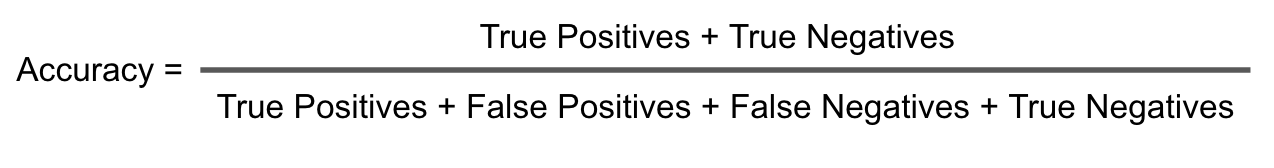
**FPR**

The percentage of mistaken negative predictions out of all the actual negative samples.

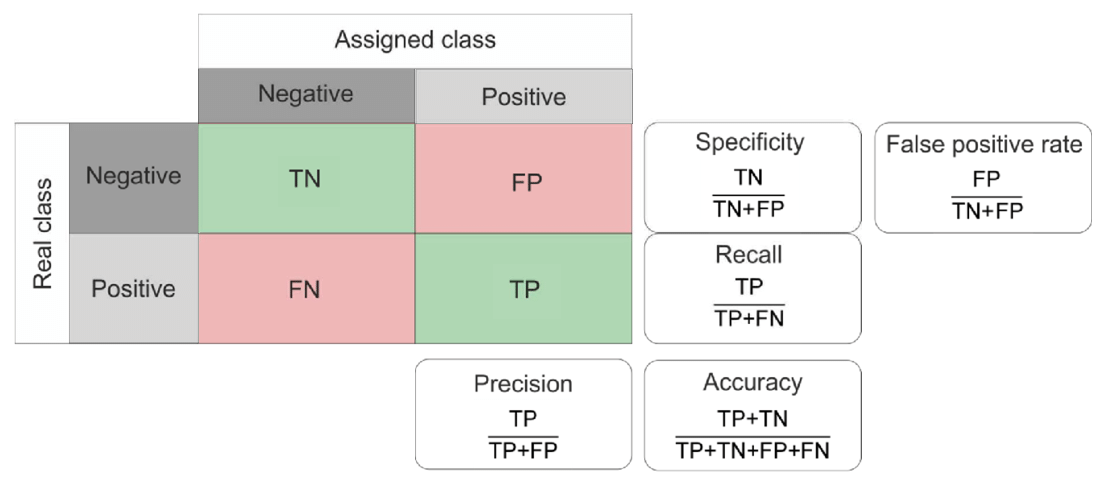


**Accuracy**

The percentage of correct predictions out of all the data.



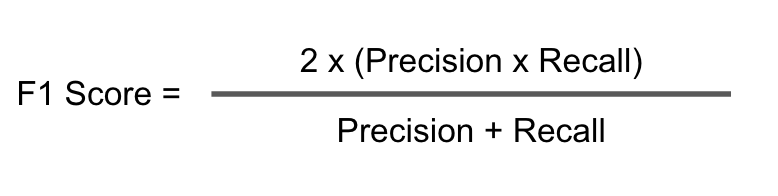
**Summary of the Confusion Matrix with its KPI's**



This visual demonstrates all the important and simple KPI's.  
It shows that FPR = 1- Specificity.  
There are a lot more KPI's, you can read about them in [Wikipedia](https://en.wikipedia.org/wiki/Confusion_matrix).

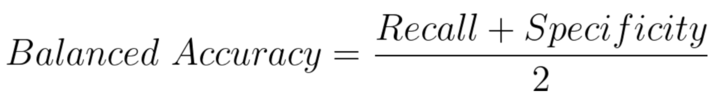
**F1 Score**

The harmonic mean of precision and recall.



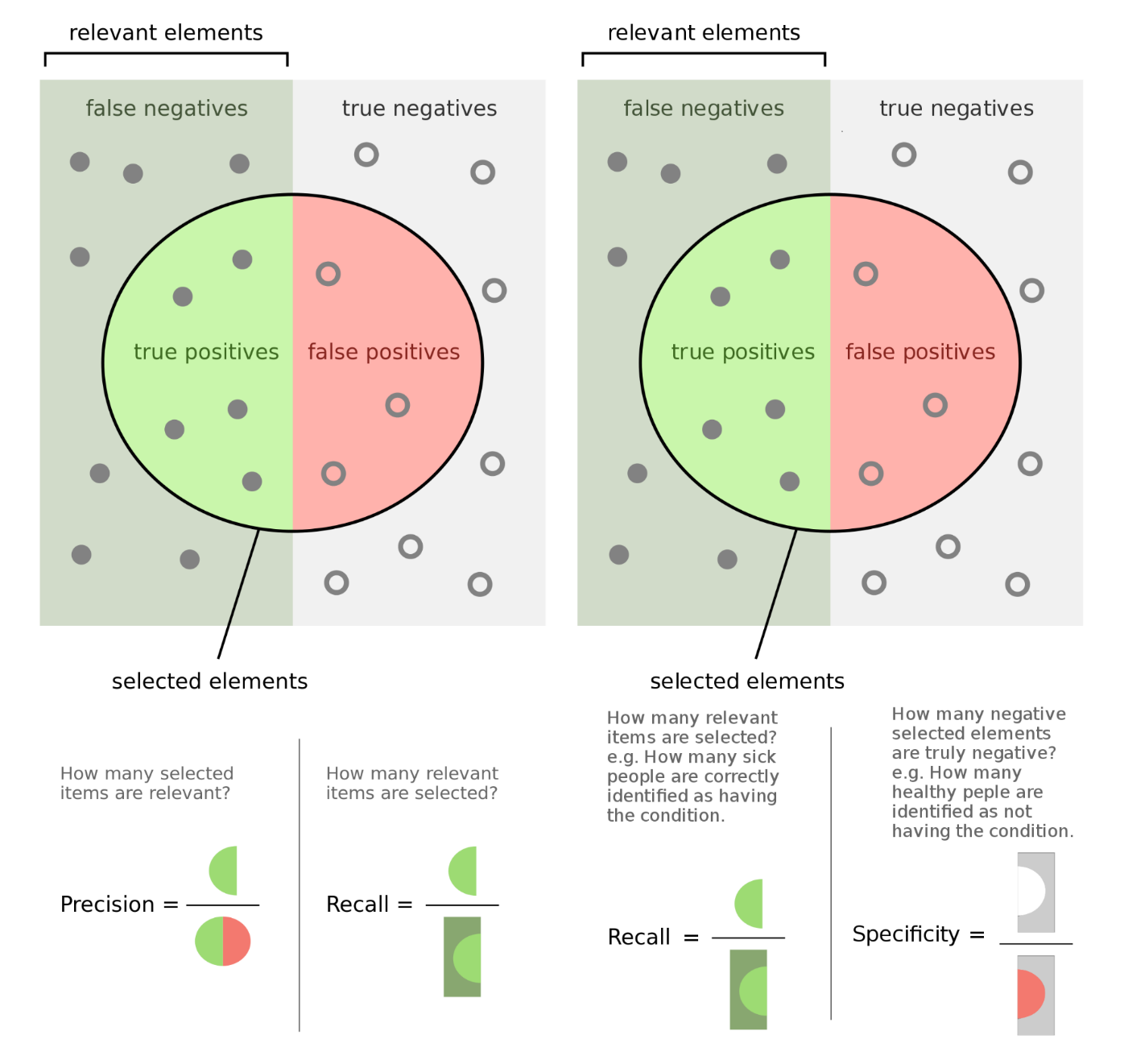
**Balanced Accuracy**

The arithmetic means of recall and specificity.



**Comparisons**

Another visual demonstrating the KPI’s:

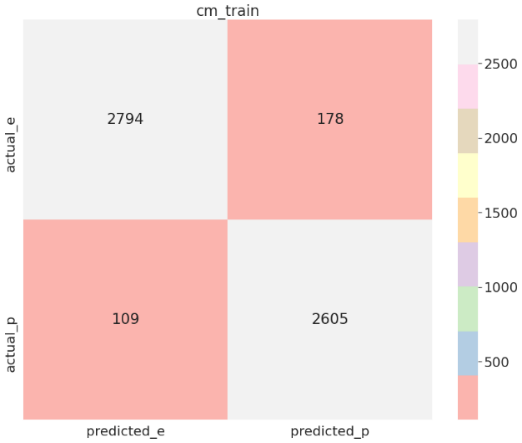
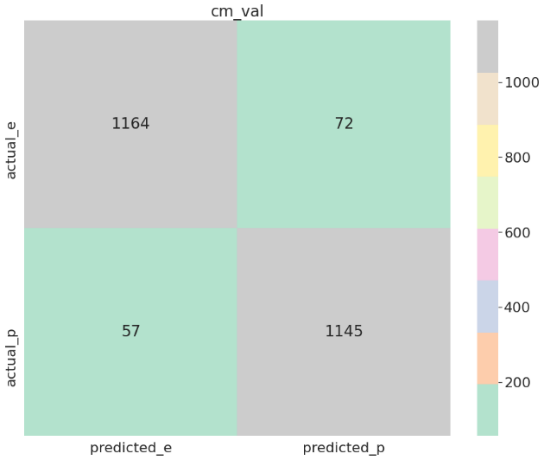


Moving back to coding, use the Scikit-learn [confusion\_matrix](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html) to calculate the confusion matrix for SGDClassifier that is trained on the Train Data with Ordinal Encoding.

Now get the tn, fp, fn and tp with NumPy [ravel](https://numpy.org/doc/stable/reference/generated/numpy.ravel.html) method.

And plot the confusion matrix with Seaborn [heatmap](https://seaborn.pydata.org/generated/seaborn.heatmap.html).

Examples:



Calculate the basic KPI's for this classifier: precision, recall, specificity, FPR and accuracy

Now use the basic KPI’s methods to calculate the more complex ones: F1 and balanced accuracy.

**Recap**: How does the classifier make its decision?

it calculates the probability (also called ‘confidence score’) for each of the labels and chooses the label with the highest confidence score.

|  | **negative probability** | **positive probability** | **prediction** |
| --- | --- | --- | --- |
| sample 1 | 0.1 | 0.9 | positive |
| sample 2 | 0.7 | 0.3 | negative |
| sample 3 | 0.4 | 0.6 | positive |

The sum of these confidence scores for each sample (each row) is 1 – since it’s a probability. In binary classification, we can use just one of them

- the score of the positive label – the negative is simple 1 minus this score.

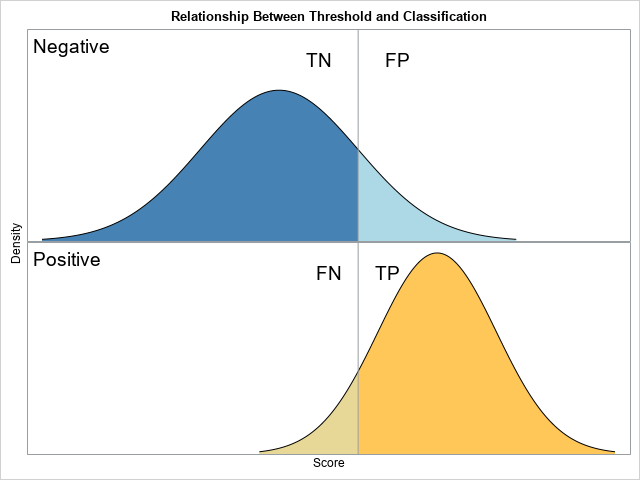
|  | **positive probability** | **prediction** |
| --- | --- | --- |
| sample 1 | 0.9 | positive |
| sample 2 | 0.3 | negative |
| sample 3 | 0.6 | positive |

If this score is higher than 0.5, the sample is predicted as positive.  
If this score is lower than 0.5, the sample is predicted as negative.

But must it be this way?

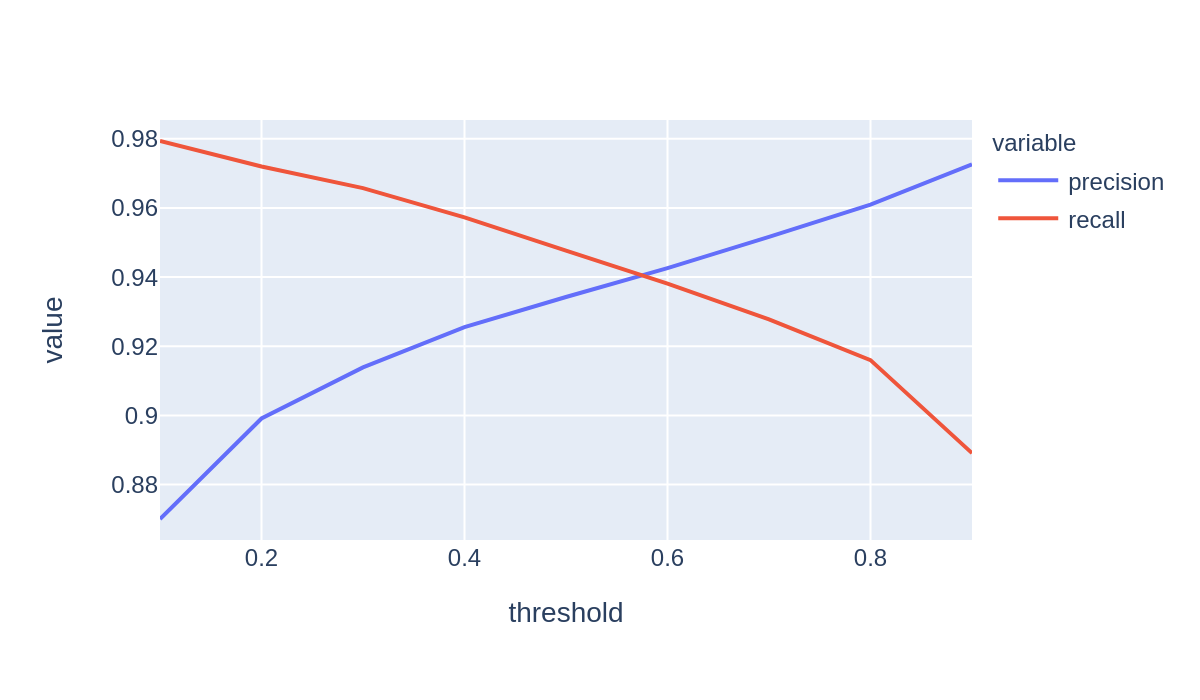
Not really! In this case, we just **chose** the threshold 0.5.

Look at the graph below: The threshold is the vertical line, that can move and change the areas of false positive (FP) and false negatives (FN).  
If we increase the threshold (move it right), a sample is predicted positive only if our confident score is high (less fp and more fn -> high precision and low recall).  
If we reduce the threshold (move it left), a sample is predicted positive even if our confident score is low (more fp and less fn -> high recall and low precision).



Check that out by plotting a graph showing the precision and recall as a function of the threshold.

My example:



There is another graph that shows our model behavior based on thresholds.  
This graph is called [ROC (Receiver Operating Characteristic)](https://en.wikipedia.org/wiki/Receiver_operating_characteristic) and it is a graph of recall vs. FPR.



When the threshold is moving from right to left, the FPR is getting bigger (more FP) and the recall is getting bigger too (less FN).  
So the model is good if the recall is high even when the FPR is low.  
The AUC (Area Under the Curve), puts this property in a single number:

The bigger the AUC, the better the model performance.

Plot the ROC with Sciking-learn [roc\_curve](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html) and calculate the AUC with Scikit-learn [roc\_auc\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score).

**More Information**

Explanation about edible and poisonous mushrooms:  
[3 edible mushrooms that are easy to find – and how to avoid the poisonous ones](https://inhabitat.com/3-edible-mushrooms-that-are-easy-to-find-and-how-to-avoid-the-poisonous-ones/)

Explanation about mushroom species:  
[Edible and Poisonous Species of Coastal BC and the Pacific Northwest](https://www.zoology.ubc.ca/~biodiv/mushroom/index.html)

Pandas-Profiling Documentation:  
[Pandas Profiling Introduction](https://pandas-profiling.github.io/pandas-profiling/docs/master/rtd/pages/introduction.html)

Guide to categorical encoding:  
[Guide to Encoding Categorical Values in Python](https://pbpython.com/categorical-encoding.html)

Explanation about KPI's:  
[4 things you need to know about AI: accuracy, precision, recall and F1 scores](https://lawtomated.com/accuracy-precision-recall-and-f1-scores-for-lawyers/)

Explanation of a few types of means:  
[Arithmetic, Geometric, and Harmonic Means for Machine Learning](https://machinelearningmastery.com/arithmetic-geometric-and-harmonic-means-for-machine-learning/)