THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 09: More classifiers, and how well do they work – ROC and AUC

Sergei V. Kalinin

Classification

- Classification is the process of identifying and grouping objects or ideas into predetermined categories.

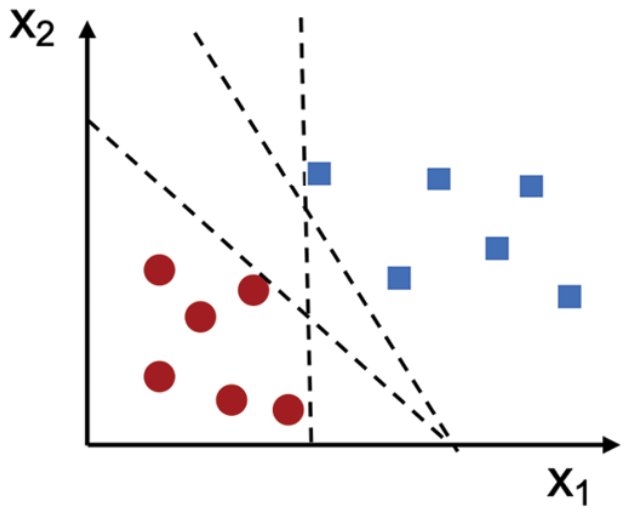
- In machine learning (ML), classification is used in predictive modeling to assign input data with a class label.

We already have several tools in our toolbox:

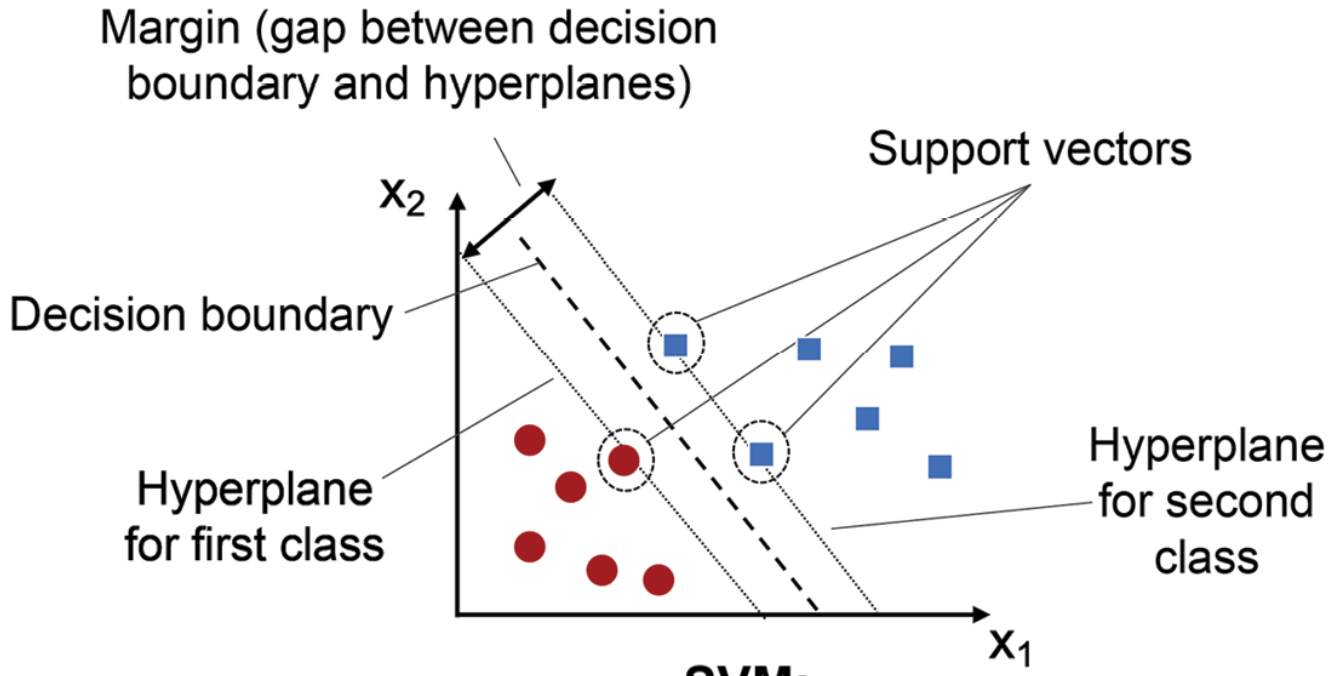
- Classifier and Regression Trees (CART), Random Forest
- Perceptron
- Adaline
- Logistic regression

There are more classifiers in scikit-learn – and all of them use the same Pythonic methods .fit(), .predict(), and (many) yield probabilities

Support Vector Machines



Which hyperplane?



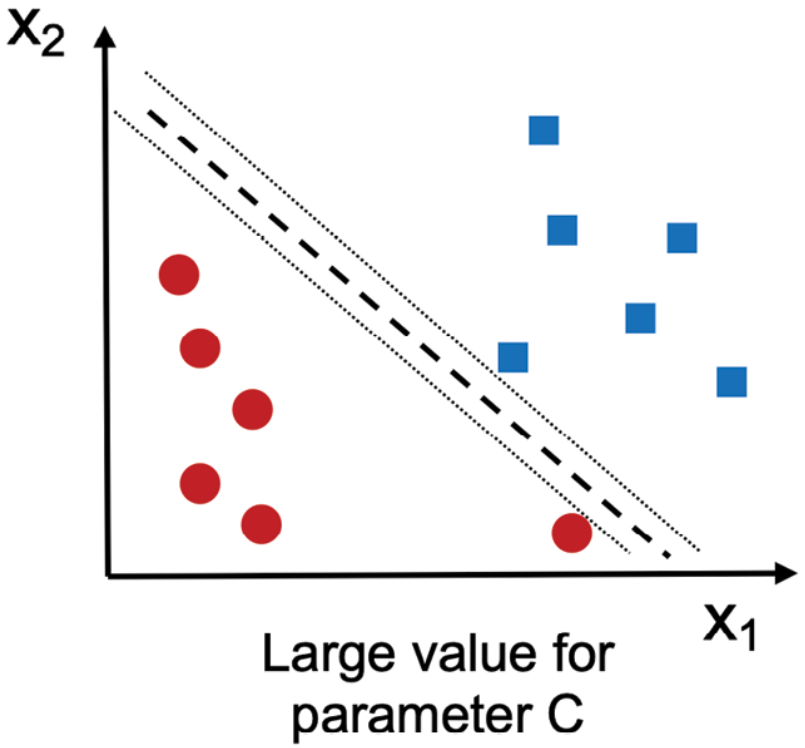
SVM:
Maximize the margin

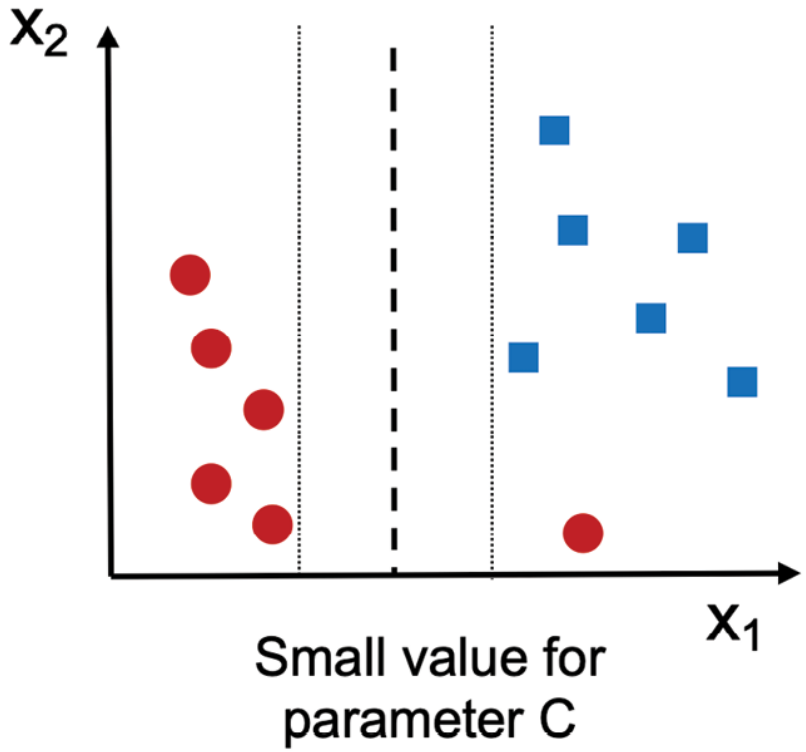
Perceptron: minimize misclassification errors.
SVM: maximize the margin.

The margin is the distance between the separating hyperplane (decision boundary) and the training examples that are closest to this hyperplane, which are called **support vectors**.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Regularization in SVMs

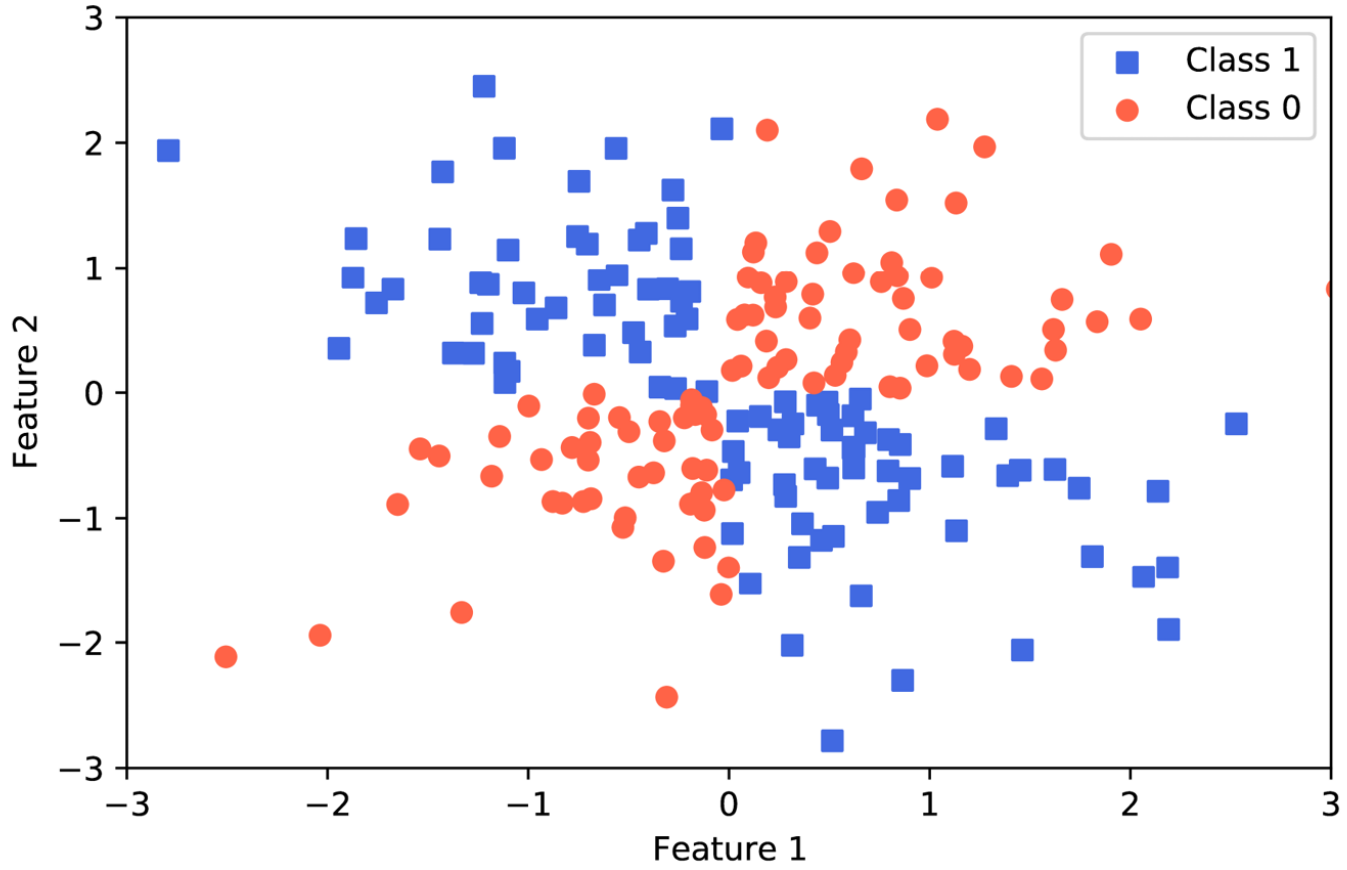




Large values of *C* correspond to large error penalties, whereas we are less strict about misclassification errors if we choose smaller values for *C*.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

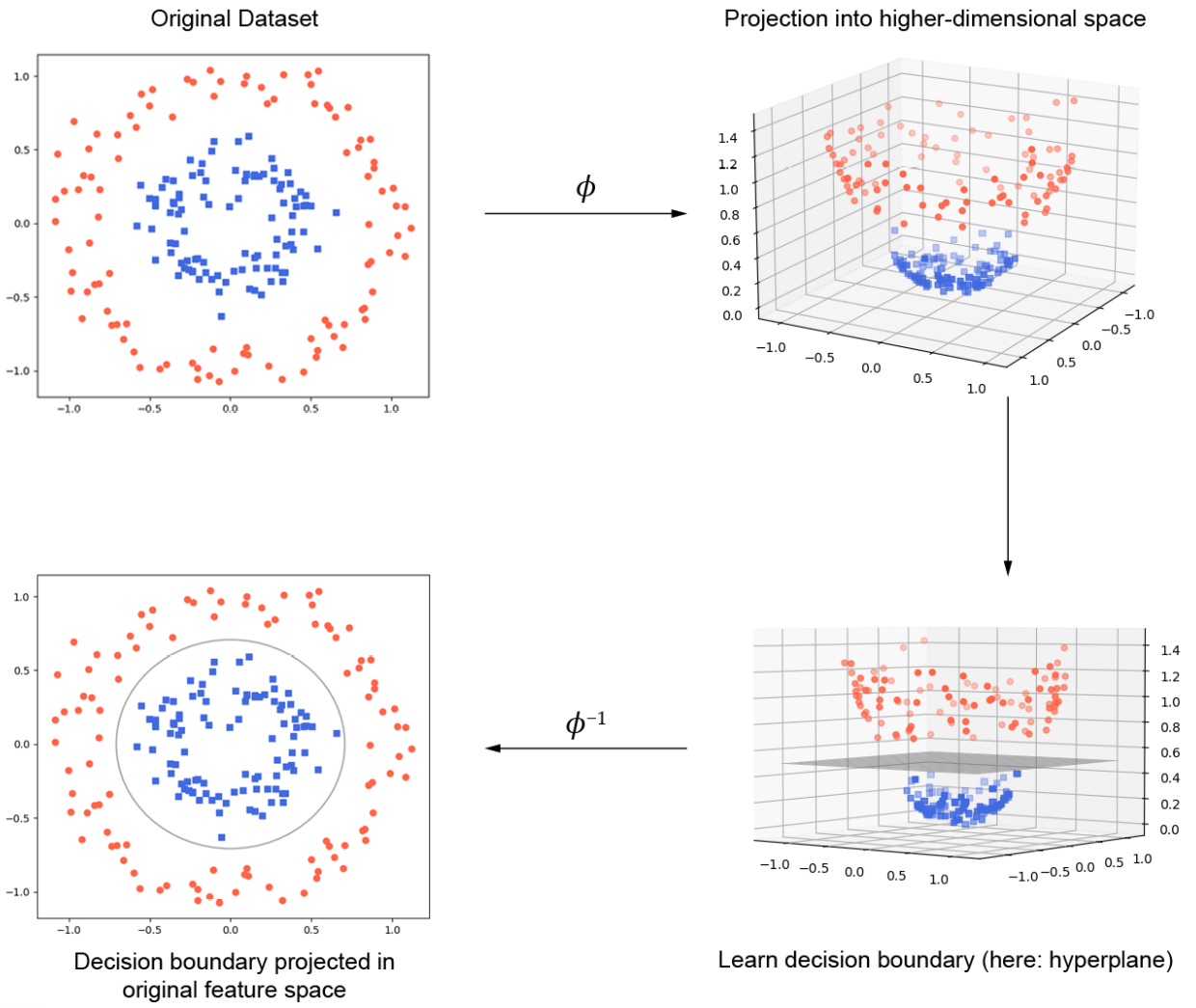
Kernel SVM: Motivation



Non-linearly separable problem

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Kernel SVM



The idea behind **kernel methods** for dealing with linearly inseparable data is to create nonlinear combinations of the original features to project them onto a higher-dimensional space via a mapping function, ϕ , where the data becomes linearly separable.

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Kernel SVM

- To train an SVM, in practice, we need to replace the dot product $\boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$ by $\phi\left(\boldsymbol{x}^{(i)}\right)^T \phi\left(\boldsymbol{x}^{(j)}\right)$.

- To save the expensive step of calculating this dot product between two points explicitly, we define a **kernel function** $\kappa\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \phi\left(\boldsymbol{x}^{(i)}\right)^T \phi\left(\boldsymbol{x}^{(j)}\right)$
-
- One of the most widely used kernels is the **radial basis function** (**RBF**) kernel, or the **Gaussian kernel**:

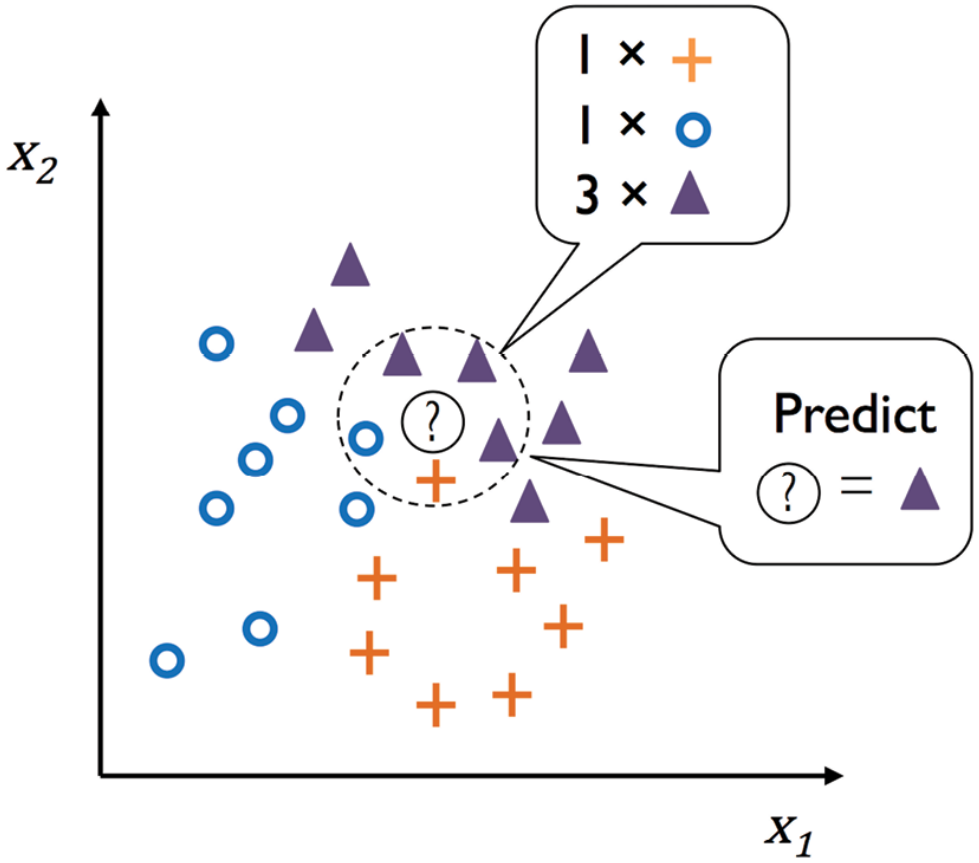
$$\kappa\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \exp\left(-\gamma\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right)$$

γ is a free parameter to be optimized.

- Roughly speaking, the term "kernel" can be interpreted as a **similarity function** between a pair of examples.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

k Nearest Neighbours (kNN) Classifier



1. Choose the number of *k* and a distance metric

2. Find the *k*-nearest neighbors of the data record that we want to classify

3. Assign the class label by majority vote

KNN is an example of a **lazy learner**. It is called "lazy" not because of its apparent simplicity, but because it doesn't learn a discriminative function from the training data but memorizes the training dataset instead.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Parametric vs. Non-parametric methods

Parametric models: we estimate parameters from the training dataset to learn a function that can classify new data points without requiring the original training dataset anymore. Typical examples of parametric models are the perceptron, logistic regression, and the linear SVM.

Non-parametric models can't be characterized by a fixed set of parameters, and the number of parameters changes with the amount of training data. Two examples of non-parametric models are the decision tree classifier/random forest and the kernel (but not linear) SVM.

KNN belongs to a subcategory of non-parametric models described as **instance-based learning**. Models based on instance-based learning are characterized by memorizing the training dataset, and lazy learning is a special case of instance-based learning that is associated with no (zero) cost during the learning process.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Pros and Cons of Memory Based Approaches

- The main advantage of memory-based approach is that the classifier immediately adapts as we collect new training data

- The downside is that the computational complexity for classifying new examples grows linearly with the number of examples in the training dataset in the worst-case scenario—unless the dataset has very few dimensions (features) and the algorithm has been implemented using efficient data structures.

- These include k-d tree (https://en.wikipedia.org/wiki/K-d_tree) and ball tree (https://en.wikipedia.org/wiki/Ball_tree), which are both supported in scikit-learn. Furthermore, next to computational costs for querying data, large datasets can also be problematic in terms of limited storage capacities.

- However, in many cases when we are working with relatively small to medium-sized datasets, memory-based methods can provide good predictive and computational performance and are thus a good choice for approaching many real-world problems.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Distances and Neighbours

- We need to choose a distance metric that is appropriate for the features in the dataset.
- Usual choice: Minkowski distance, a generalization of the Euclidean (p = 2) and Manhattan (p = 1) distance:

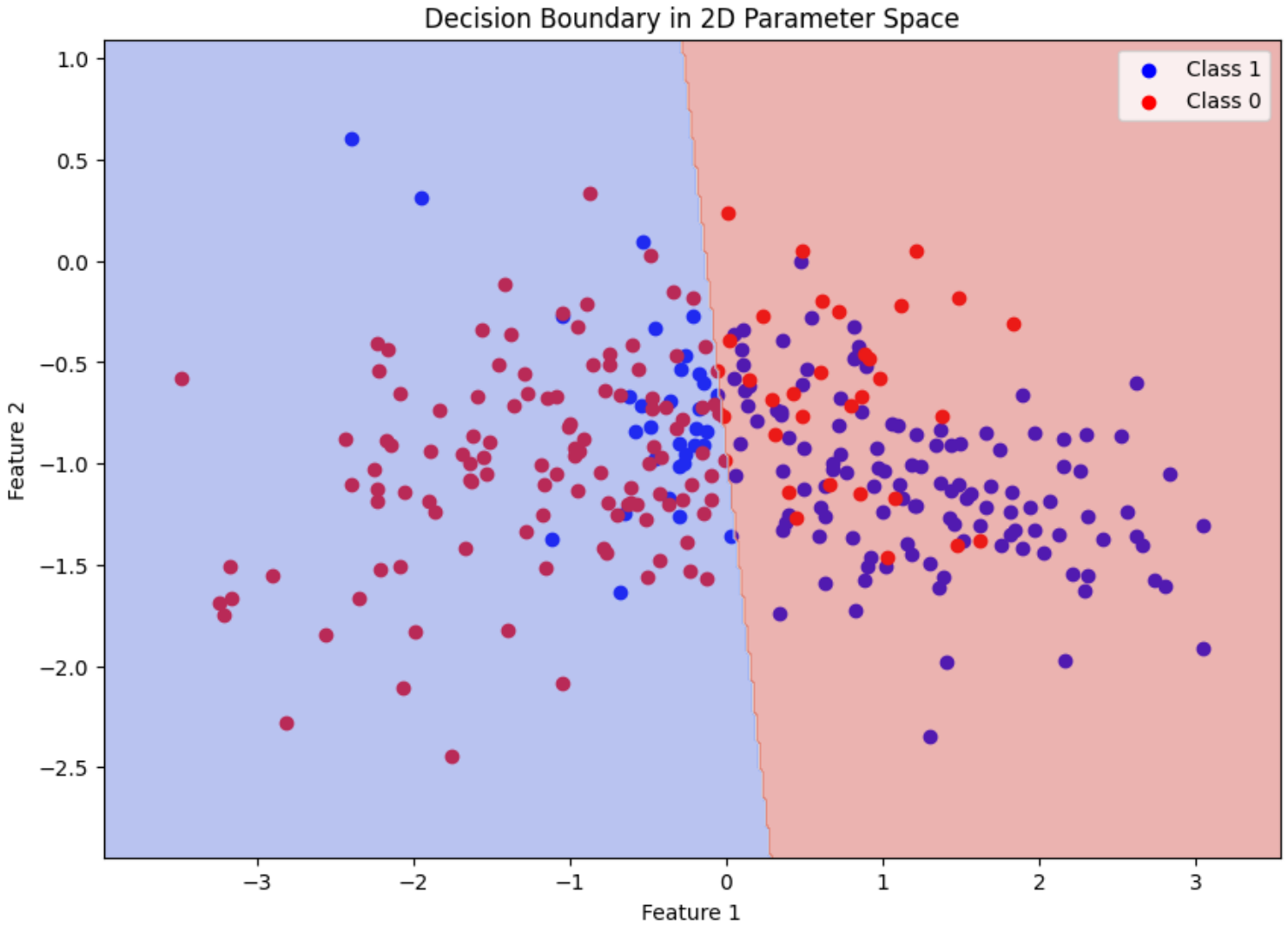
$$d(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \sqrt[p]{\sum_k \left| x_k^{(i)} - x_k^{(j)} \right|^p}$$

- Many other distance metrics are available in scikit-learn and can be provided to the metric parameter. They are listed at https://scikit-learn.org/stable/modules/generated/sklearn.metrics.DistanceMetric.html.
- KNN is very susceptible to overfitting due to the **curse of dimensionality**, where feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset.
- For regression and SVM, we use regularization to avoid this problem.
- We cannot use regularization for decision trees and KNN. Instead, we can use feature selection and dimensionality reduction techniques

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

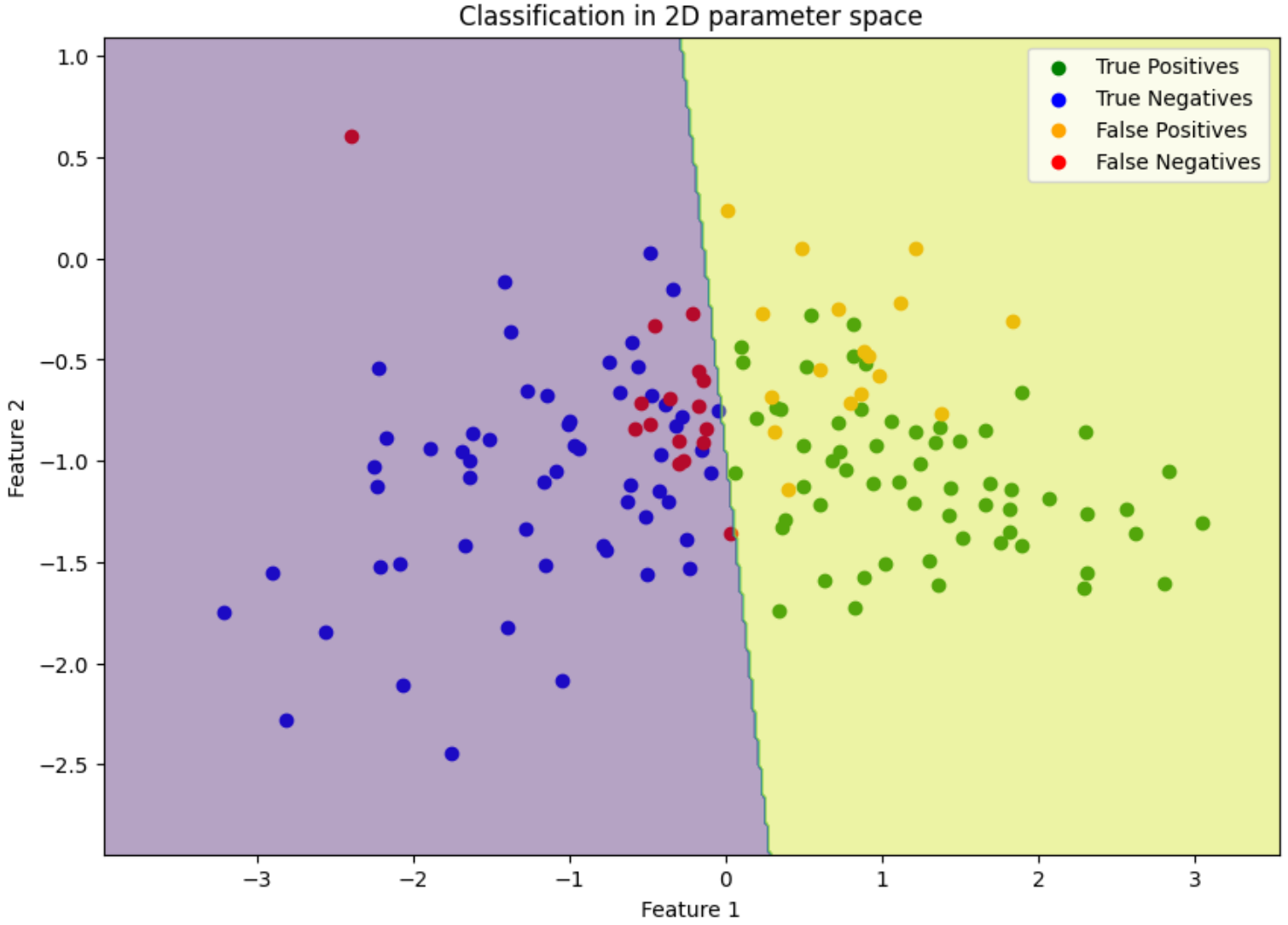
How well do classifiers work?

Let’s consider the binary classifier



- Classification is not perfect
- We have examples of Class 1 identified as 0, and Class 0 as 1

Let’s consider the binary classifier



- Does the mis-identification matter?

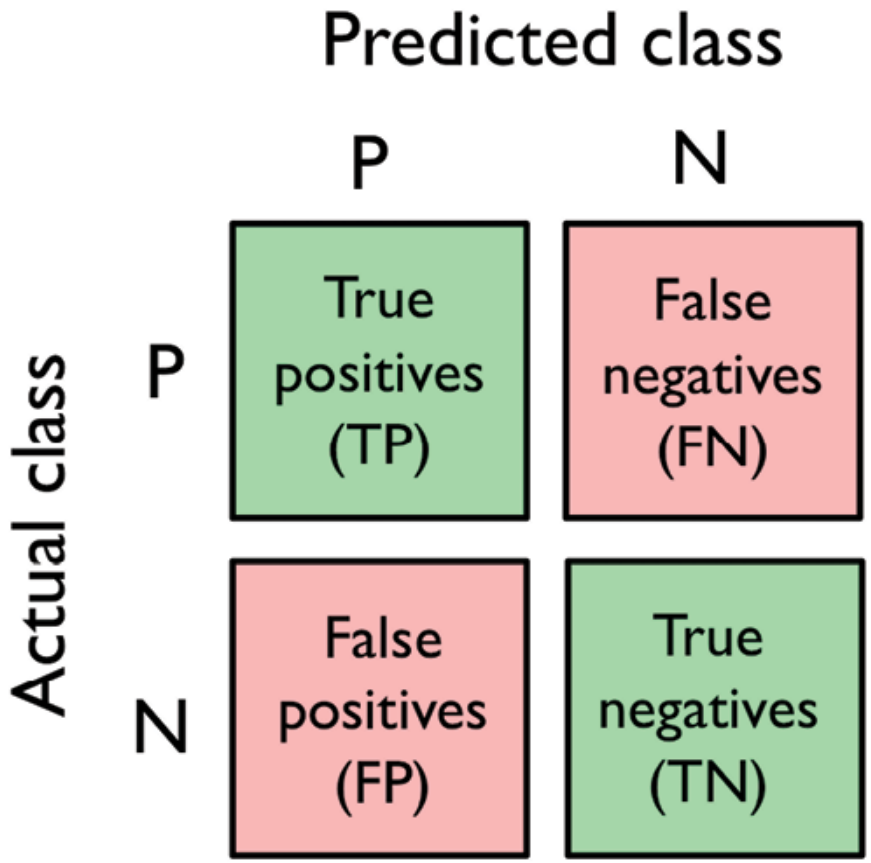
It depends on the problem!

- Molecular prediction: toxicity
- Medical tests: tumor, COVID, flu, etc…
- Finance: fraud detection
- Metal detector: sensitivity threshold
- Object detection: face identification
- Quality control: ripe/rotten

Standard assumptions

- Standard Cost Model
  - correct classification costs 0
  - cost of misclassification depends only on the class, not on the individual example
  - over a set of examples costs are additive
- Costs or Class Distributions:
  - are not known precisely at evaluation time
  - may vary with time
  - may depend on where the classifier is deployed
- True FP and TP do not vary with time or location, and are accurately estimated.

Basic definitions



Error:

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

Accuracy:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Same definition – different names

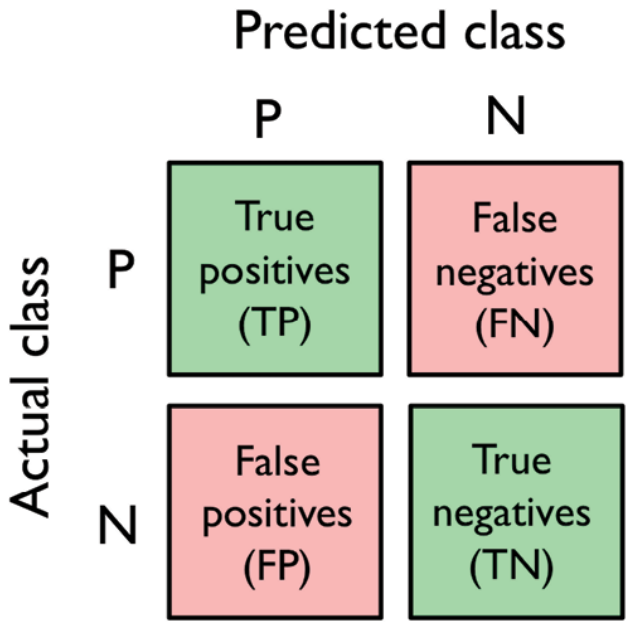
Event happens?	Forecast says event will happen?	
	Yes	No
Yes	Hit	Miss
No	False Alarm	Correct Rejection

From Simon J. Mason, International Research Institute for Climate Prediction, Columbia University

Basic definitions

False positive rate: $$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

True positive rate: $$TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$



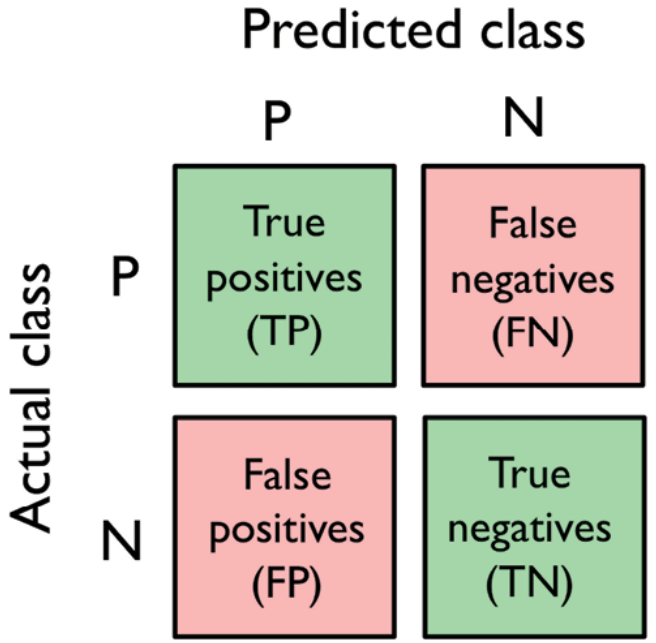
Useful for imbalanced class problems

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Basic definitions

Recall: $$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

Precision: $$PRE = \frac{TP}{TP + FP}$$



For medicine (cancer detection):

Optimizing for recall helps with minimizing the chance of not detecting a malignant tumor. However, this comes at the cost of predicting malignant tumors in patients although the patients are healthy (a high number of FPs).

Optimize for precision, on the other hand, we emphasize correctness if we predict that a patient has a malignant tumor. However, this comes at the cost of missing malignant tumors more frequently (a high number of FNs).

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

There is always more....

F1 Score:

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

David M. W. Power *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*, https://arxiv.org/abs/2010.16061.

Matthews correlation coefficient:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

D. Chicco and G. Jurman, *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation* by *BMC Genomics*. pp. 281-305, 2012, https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7.

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

… and more

		Predicted condition			
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) = TPR + TNR − 1	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)**	**True positive (TP)**, hit	**False negative (FN)**, type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate $= \frac{FN}{P} = 1 - TPR$
	Negative (N)	**False positive (FP)**, type I error, false alarm, overestimation	**True negative (TN)**, correct rejection	False positive rate (FPR), probability of false alarm, fall-out $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{PP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{PN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{PP} = 1 - PPV$	Negative predictive value (NPV) = $\frac{TN}{PN}$ = 1 − FOR	Markedness (MK), deltaP (Δp) = PPV + NPV − 1	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2PPV \times TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index = $\frac{TP}{TP + FN + FP}$

Sources: [17][18][19][20][21][22][23][24][25] view · talk · edit

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Limitation of the scalar measures

- **A scalar does not tell the whole story.**
  - There are fundamentally two numbers of interest (FP and TP), a single number invariably loses some information.
  - How are errors distributed across the classes ?
  - How will each classifier perform in different **testing** conditions (costs or class ratios other than those measured in the experiment) ?

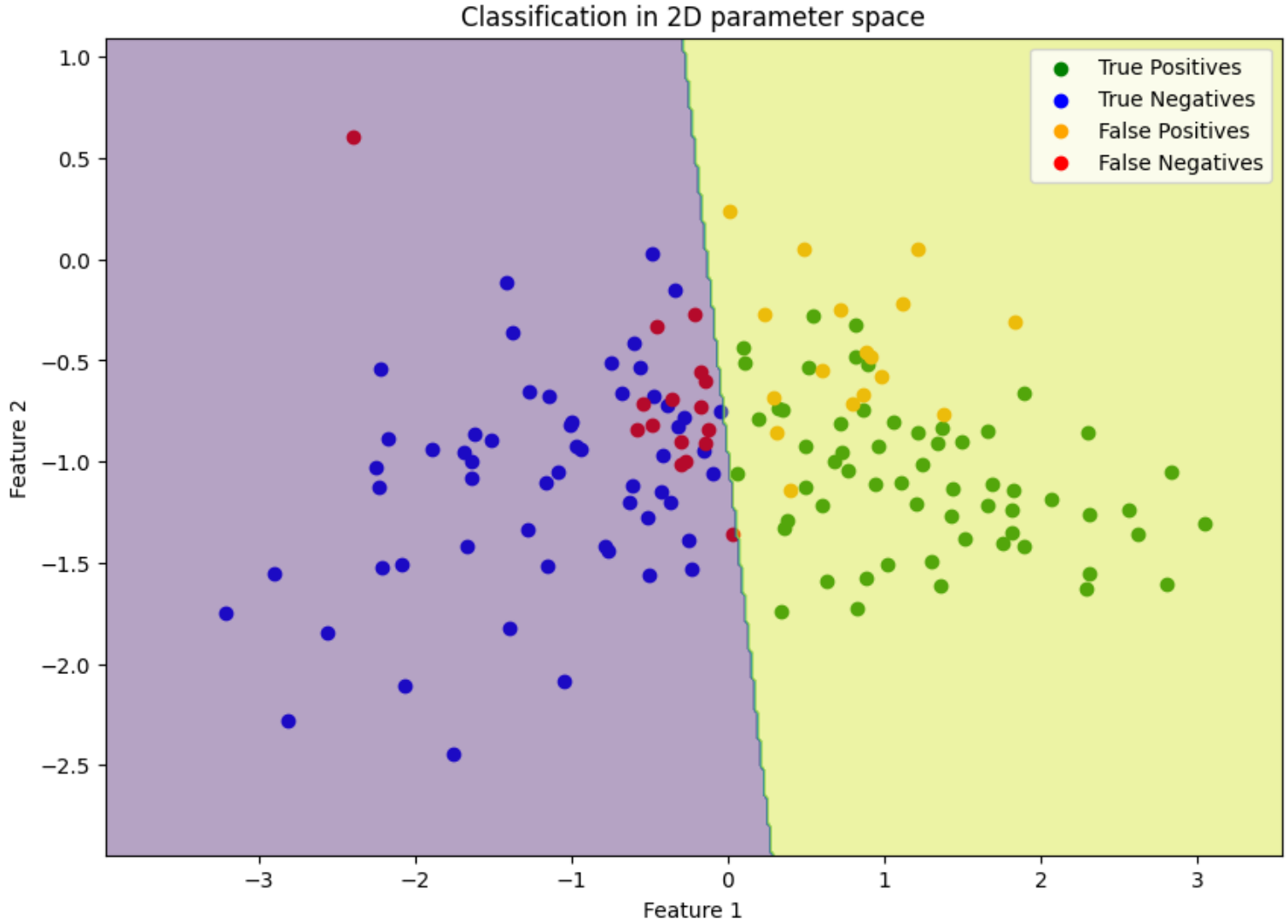
- **A scalar imposes a linear ordering on classifiers.**
  - what we want is to identify the conditions under which each is better.

Limitations of Scalar Measures

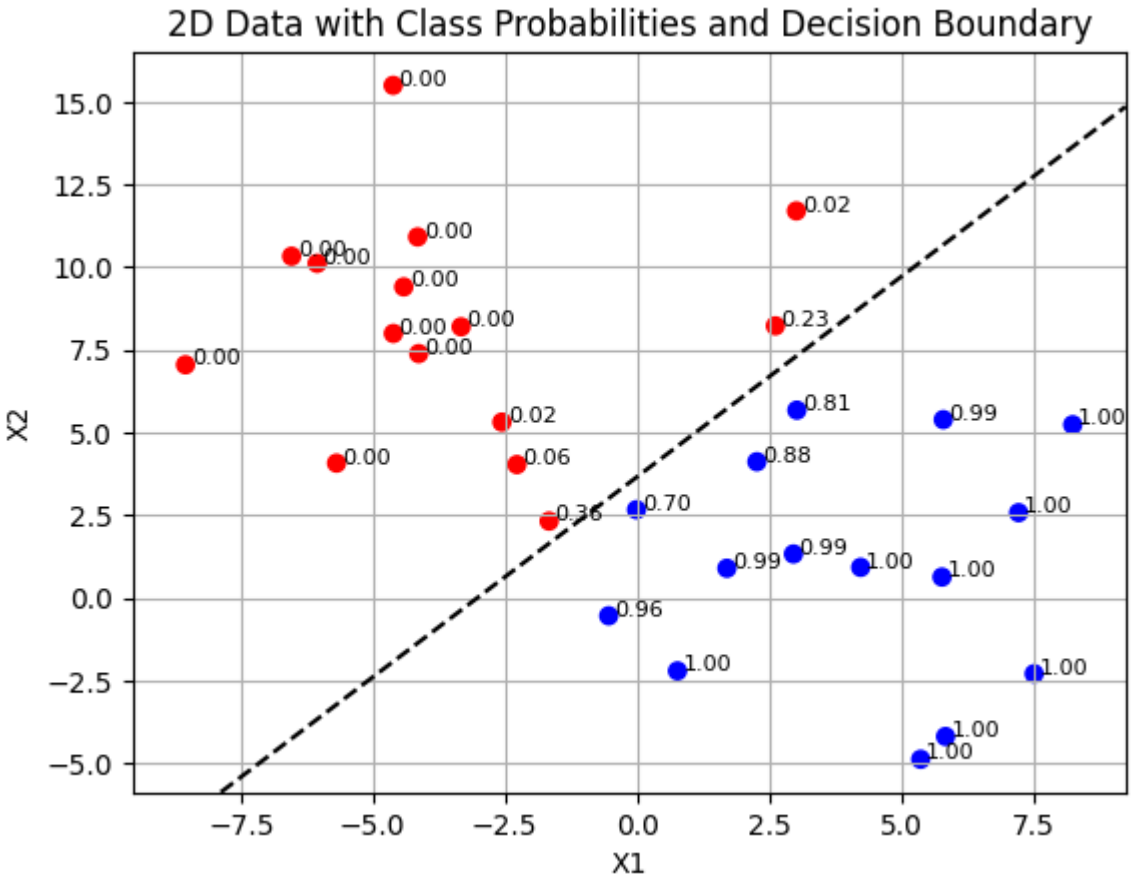
- **A table of scalars is just a mass of numbers.**
  - No immediate impact
  - Poor way to present results in a paper
  - Equally poor way for an experimenter to analyze results

- **Some scalars (accuracy, expected cost) require precise knowledge of costs and class distributions.**
  - Often these are not known precisely and might vary with time or location of deployment.

Can we tune the classifier?



- Depending on context, cost of FP and FN errors can be different
- Can we tune the classifiers to match the business case?



1. Train the Classifier on train data

2. Get Class Probabilities for test dataset. This will give the probability of each instance belonging to the positive class.

3. Order Probabilities: Sort by the predicted probabilities of belonging to the positive class.

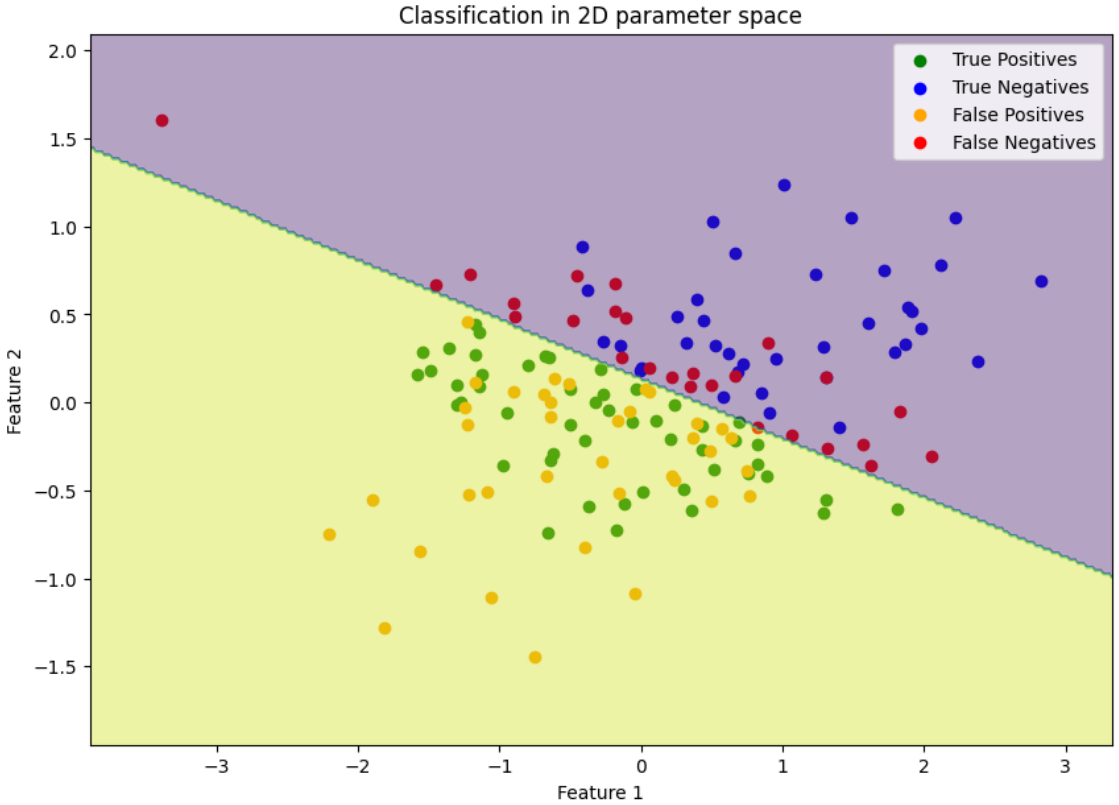
4. Calculate TPR and FPR for Each Threshold: For every unique probability value, treat it as a threshold. Instances with probabilities above (or equal to) the threshold are classified as positive, and those below are classified as negative. For each threshold, calculate TPR and FPR.

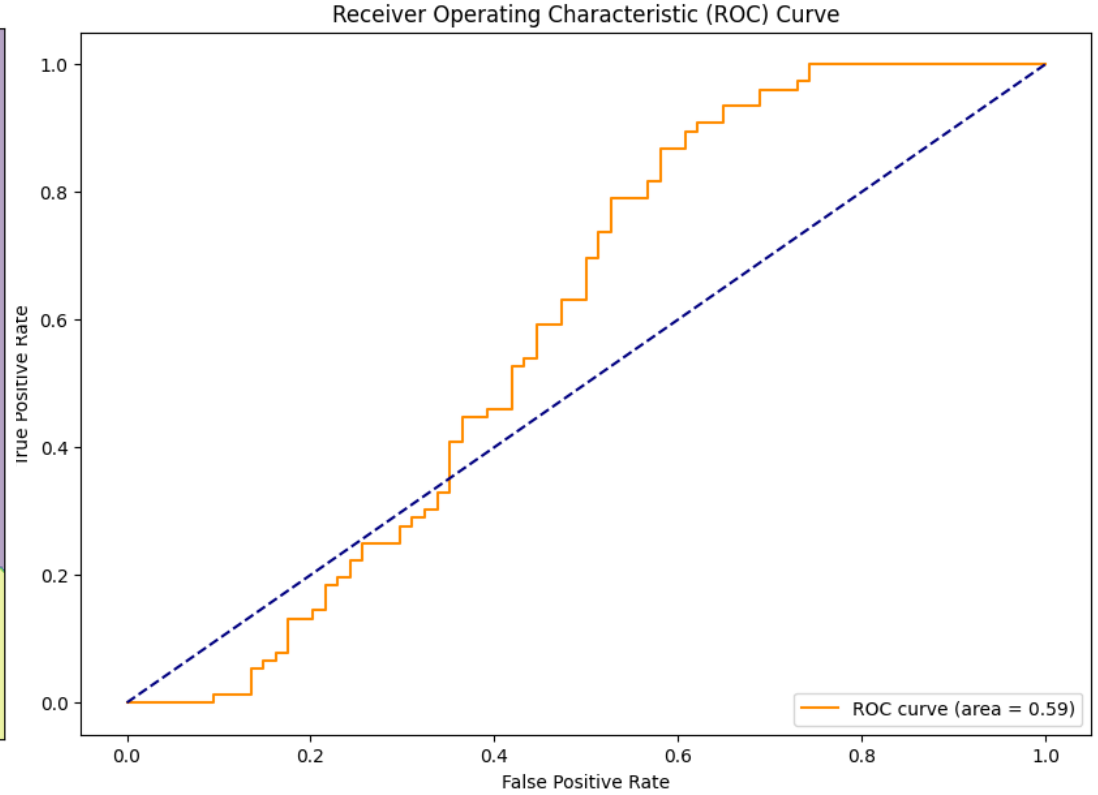
5. Plot ROC Curve: Plot TPR vs. FPR for each threshold, resulting in the ROC curve.

6. Calculate AUC: Compute the Area Under the Curve (AUC)

Receiver-Operator Characteristics (ROC)

Class separation 0

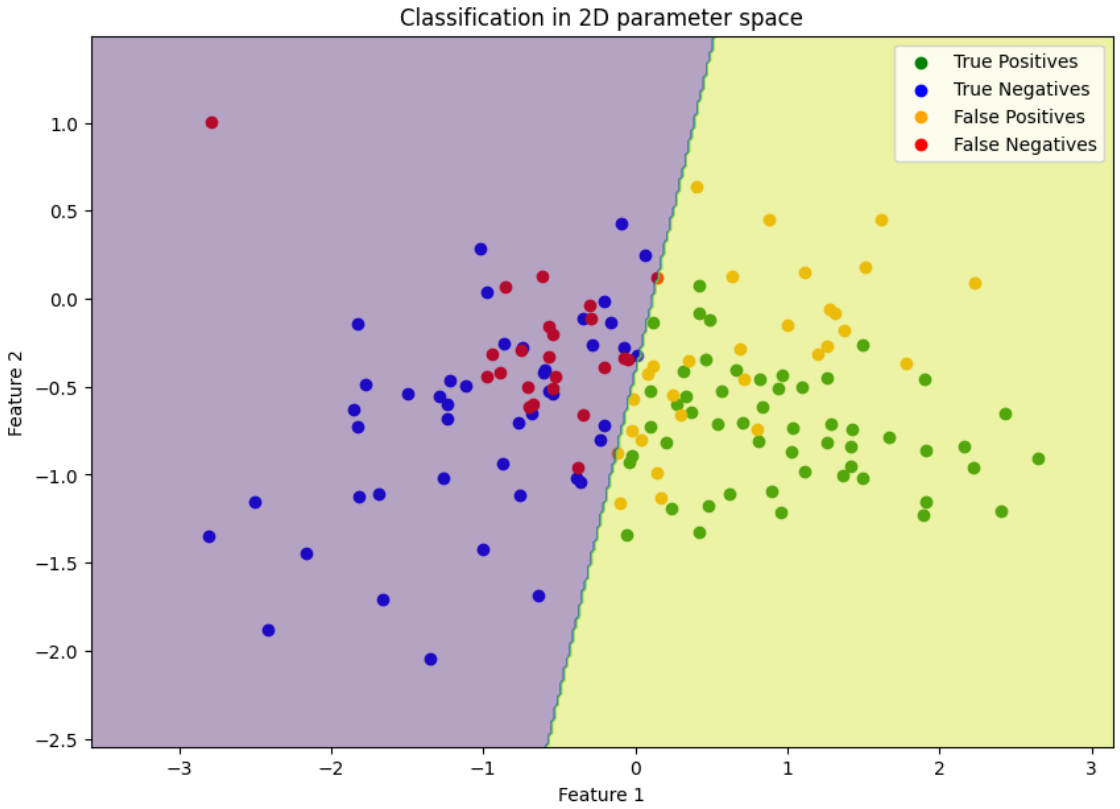


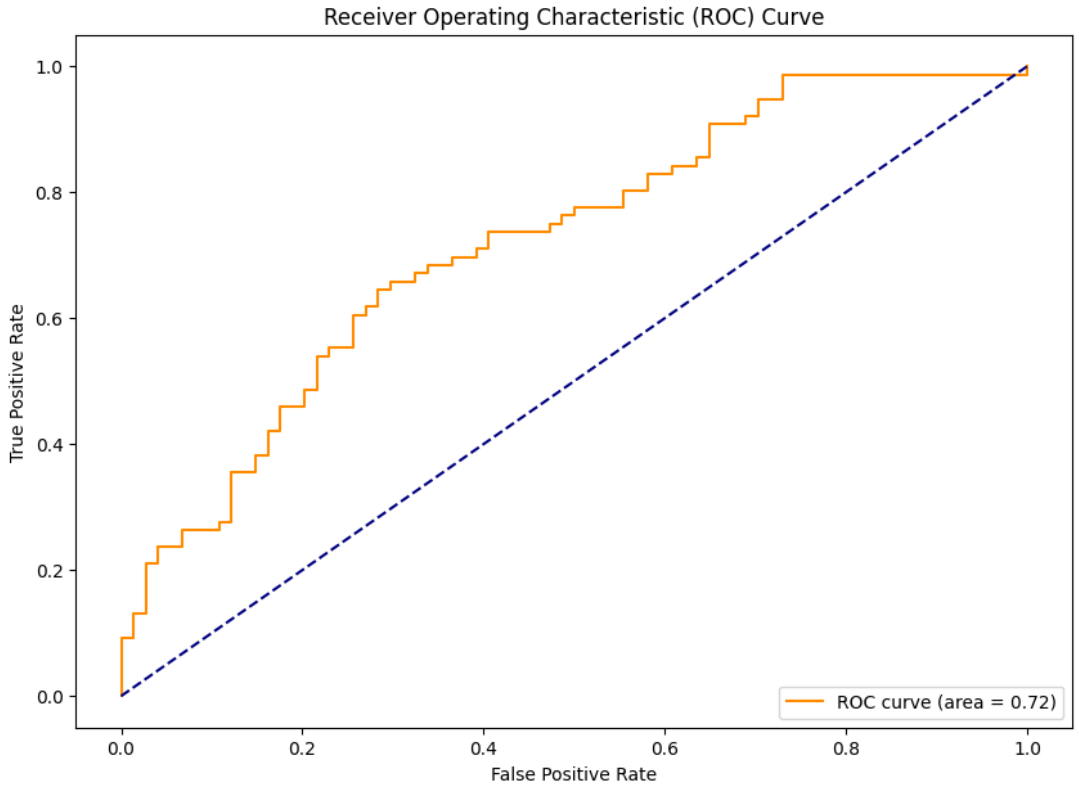


The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Receiver-Operator Characteristics (ROC)

Class separation 0.6

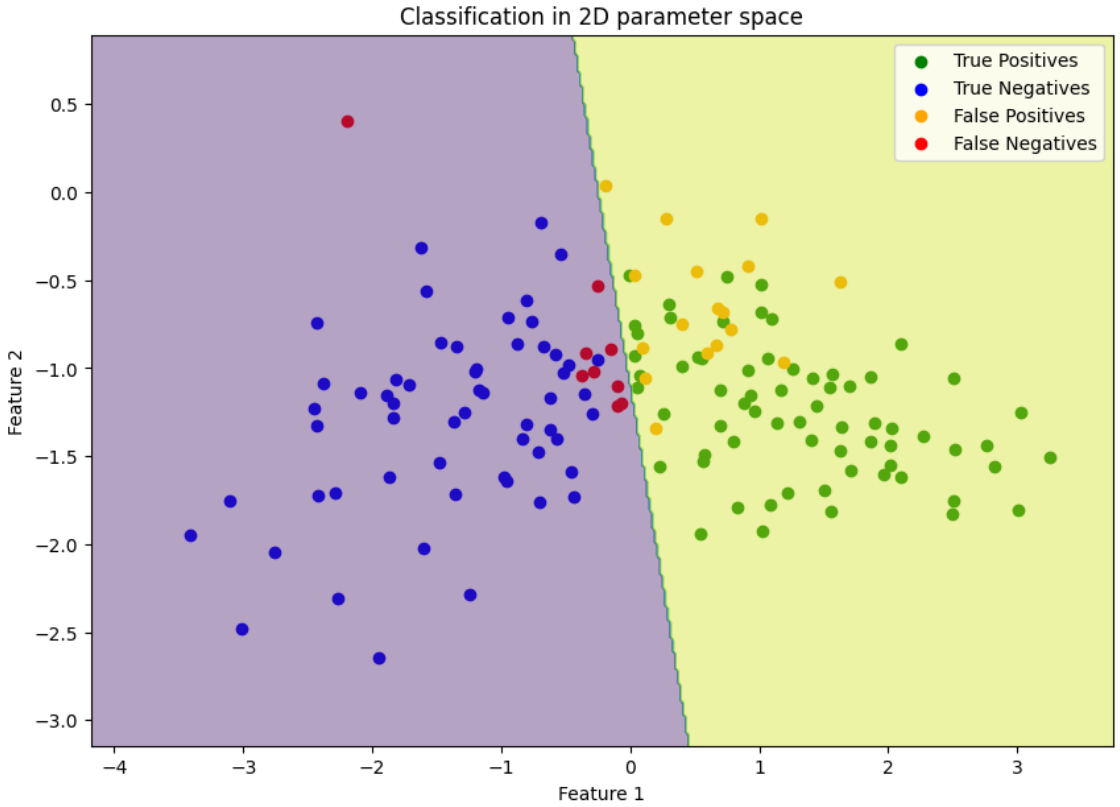


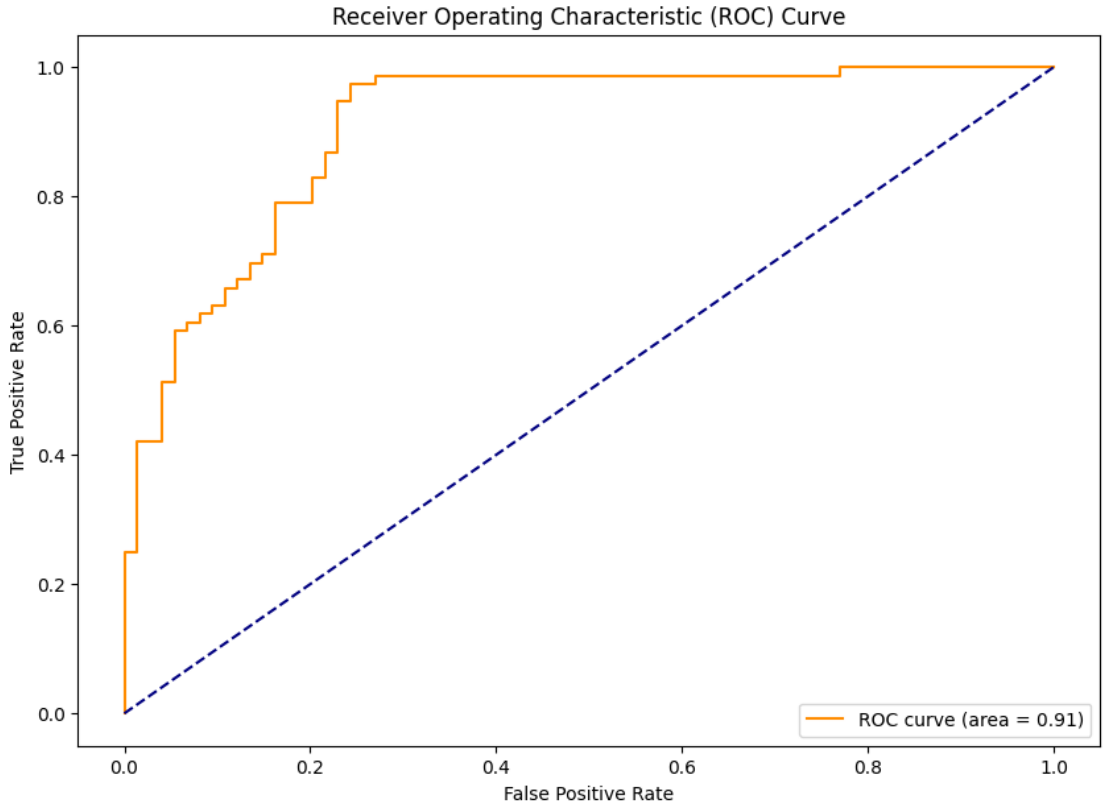


The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Receiver-Operator Characteristics (ROC)

Class separation 1.2



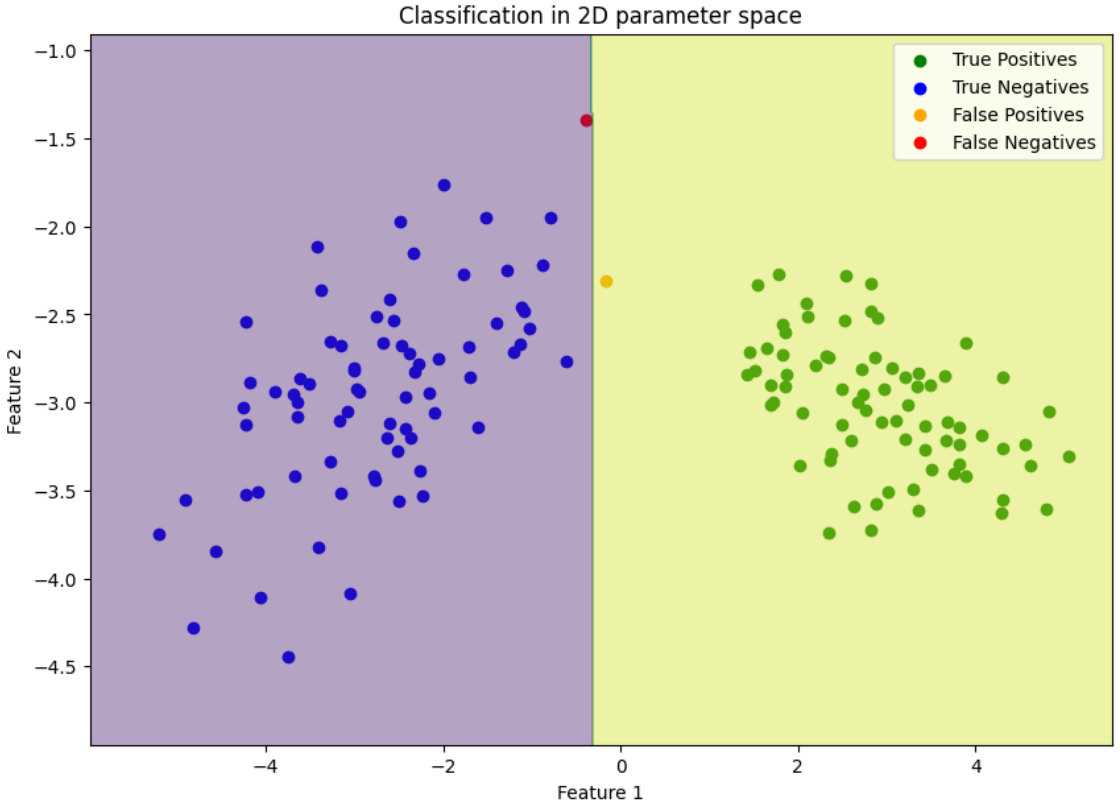


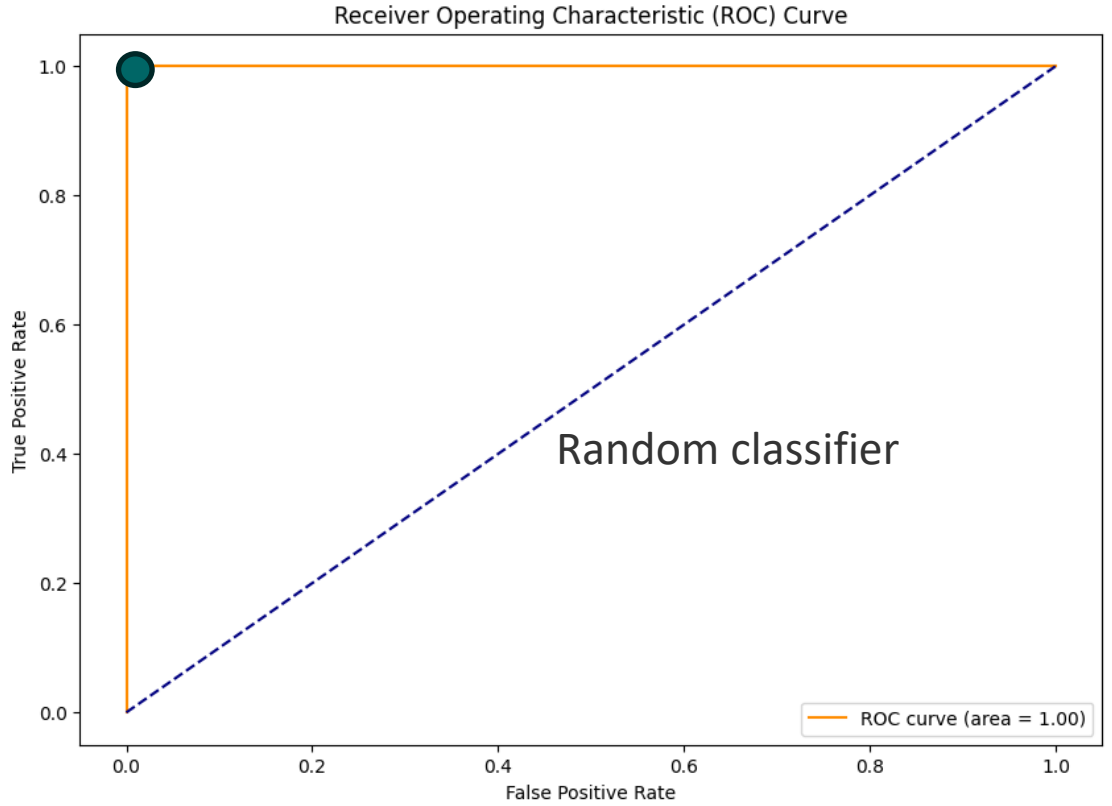
The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Receiver-Operator Characteristics (ROC)

Class separation 3

Ideal classifier





The Receiver Operating Characteristic (ROC) curve is a graphical representation used to assess the performance of a binary classifier system. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for different decision thresholds.

Advantage of ROC approach:

- A classifier produces a single ROC point.
- If the classifier has a "sensitivity" parameter, varying it produces a series of ROC points (confusion matrices).
- Alternatively, if the classifier is produced by a learning algorithm, a series of ROC points can be generated by varying the class ratio in the training set.

- ROC curves allow:
  - Visualization of all tradeoffs a model can make
  - Comparison of models across types of tradeoffs
- AUC – an aggregate measure of quality across tradeoffs
- Operating points are the tradeoff selected for specific application

From slides by Robert Holte, Computing Science Dept., University of Alberta **Cost-Sensitive Classifier Evaluation**

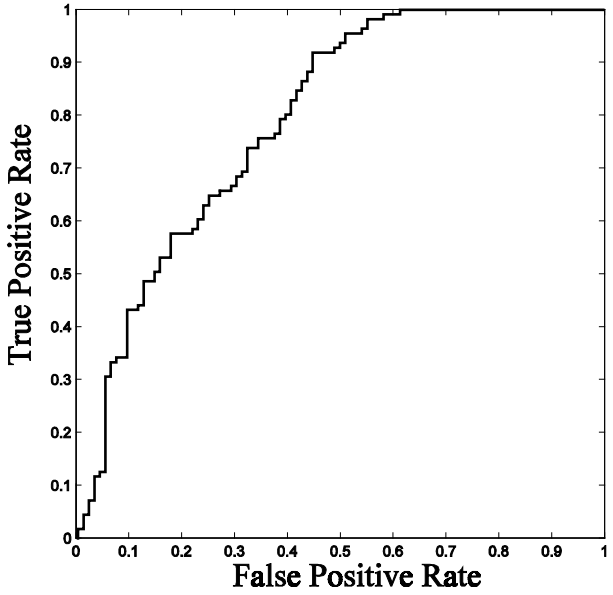
From ROC curves to Business models

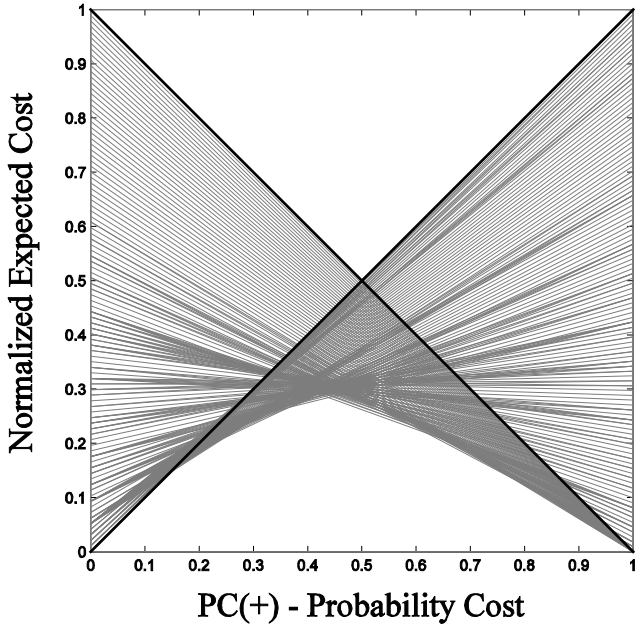
$Y = FN \bullet X + FP \bullet (1-X)$

So far, $X = p(+)$, making Y = error rate

$$X = \frac{p(+) \bullet C(-|+)}{p(+) \bullet C(-|+) + (1-p(+)) \bullet C(+|-)}$$

Y = expected cost normalized to [0,1]





Cost curves enable easy visualization of

- Average performance (expected cost)
- Operating range
- Confidence intervals on performance
- Difference in performance and its significance

From slides by Robert Holte, Computing Science Dept., University of Alberta **Cost-Sensitive Classifier Evaluation**

What if we have multiple classes?

1. One-vs-One (OvO) or One-vs-Rest (OvR) approach: This involves decomposing the multi-class classification problem into multiple binary classification problems and plotting a ROC curve for each one.

1. **One-vs-Rest (OvR)**: For K classes, you train K separate binary classifiers. For each classifier, one class is treated as positive while all others are treated as negative. We will have K separate ROC curves.

2. **One-vs-One (OvO)**: For K classes, we train a binary classifier for every pair of classes. This results in $K(K-1)/2$ classifiers and ROC curves.

For multi-class classification, we can compute the AUC for each class individually and then average them, providing a single scalar value that summarizes the performance.

What if we have multiple classes?

1. **Macro-averaging and Micro-averaging**:

    1. **Macro-averaging**: Calculate the ROC curve for each class and then average them to get a single ROC curve.

    2. **Micro-averaging**: Aggregate the outcomes of individual classifications (over all classes) to compute the ROC curve. This means you'll combine all the true positives, false positives, true negatives, and false negatives across all classes and then compute the TPR and FPR.

What if we have imbalanced classes?

- Assign a larger penalty to wrong predictions on the minority class. Via scikit-learn, adjusting such a penalty is as convenient as setting the *class_weight* parameter to *class_weight*='balanced', which is implemented for most classifiers
- Upsampling the minority class or downsampling the majority class
- Generation of synthetic training examples

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn