Artificial Intelligence and Machine Learning

Unit II

Model Selection and Evaluation Metrics

My own latex definitions

In [2]:
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
plt.style.use('seaborn-whitegrid')

font = {'family' : 'Times',
        'weight' : 'bold',
        'size'   : 12}

matplotlib.rc('font', **font)


# Aux functions

def plot_grid(Xs, Ys, axs=None):
    ''' Aux function to plot a grid'''
    t = np.arange(Xs.size) # define progression of int for indexing colormap
    if axs:
        axs.plot(0, 0, marker='*', color='r', linestyle='none') #plot origin
        axs.scatter(Xs,Ys, c=t, cmap='jet', marker='.') # scatter x vs y
        axs.axis('scaled') # axis scaled
    else:
        plt.plot(0, 0, marker='*', color='r', linestyle='none') #plot origin
        plt.scatter(Xs,Ys, c=t, cmap='jet', marker='.') # scatter x vs y
        plt.axis('scaled') # axis scaled

def linear_map(A, Xs, Ys):
    '''Map src points with A'''
    # [NxN,NxN] -> NxNx2 # add 3-rd axis, like adding another layer
    src = np.stack((Xs,Ys), axis=Xs.ndim)
    # flatten first two dimension
    # (NN)x2
    src_r = src.reshape(-1,src.shape[-1]) #ask reshape to keep last dimension and adjust the rest
    # 2x2 @ 2x(NN)
    dst = A @ src_r.T # 2xNN
    #(NN)x2 and then reshape as NxNx2
    dst = (dst.T).reshape(src.shape)
    # Access X and Y
    return dst[...,0], dst[...,1]


def plot_points(ax, Xs, Ys, col='red', unit=None, linestyle='solid'):
    '''Plots points'''
    ax.set_aspect('equal')
    ax.grid(True, which='both')
    ax.axhline(y=0, color='gray', linestyle="--")
    ax.axvline(x=0, color='gray',  linestyle="--")
    ax.plot(Xs, Ys, color=col)
    if unit is None:
        plotVectors(ax, [[0,1],[1,0]], ['gray']*2, alpha=1, linestyle=linestyle)
    else:
        plotVectors(ax, unit, [col]*2, alpha=1, linestyle=linestyle)

def plotVectors(ax, vecs, cols, alpha=1, linestyle='solid'):
    '''Plot set of vectors.'''
    for i in range(len(vecs)):
        x = np.concatenate([[0,0], vecs[i]])
        ax.quiver([x[0]],
                  [x[1]],
                  [x[2]],
                  [x[3]],
                  angles='xy', scale_units='xy', scale=1, color=cols[i],
                  alpha=alpha, linestyle=linestyle, linewidth=2)
```

Recap previous lecture

- Decision Trees and drawbacks
- Bootstrap and Bagging
- Random Forest and their advantages
- Application to Xbox Human Pose Estimation

Today's lecture

It is about how to we evaluate models

1) Model selection and Cross-Validation

2) Hyper-parameter tuning

3) Metrics for Evaluation (mainly for classification)

This lecture material is taken from

- Stanford class
- Stanford slides
- Stanford notes
- Tibishrinar - Chapter 7 page 219
- Sklearn model selection
- Cimi book chapter 5

ERM and Its Limits: Bias-Variance Trade-off

$$\colorbox{yellow}{\$}\epsilon = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(y_n, f\left(x_n\right)\right)\$$$

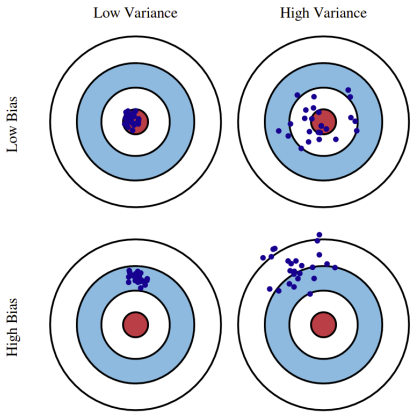
The generalization error (test error) of a ML system can be always decomposed into **two parts**:

- **Bias Error**
- **Variance Error**
- **Noise Term (depends on the data only)**

Bias-Variance Trade-off

Bias-Variance Tradeoff as Dartboard

- Each dart shot below is a training experiment!



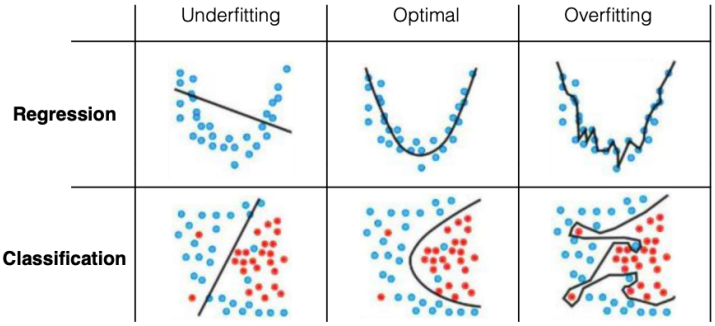
BIAS-Variance Trade-off

- The **bias error** is produced by weak assumptions in the learning algorithm
  - **High bias** can cause an algorithm to **miss the relevant relations between features and target outputs**
  - Problem know as `underfitting`. Solution: increase the complexity/expressiveness of your ML algorithm!

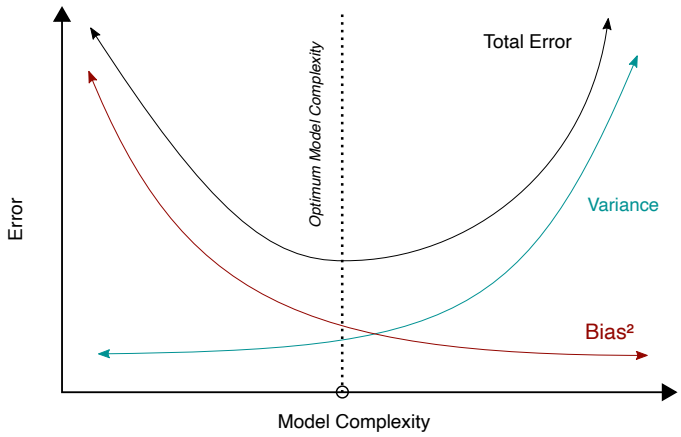
Bias-VARIANCE Trade-off

- The **variance** is an error produced by an **oversensitivity to small fluctuations in the training set**
  - High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs
  - Problem know as `overfitting`. Solution: decrease the model complexity or add strong regularization.

Over or Under Fitting



Error in function of model complexity



Bias-Variance Error "Proof Sketch"

We have a finite number of samples $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$ that are generated from a function f plus error so $y = f(\mathbf{x}) + \epsilon$

- $\mathbb{E}[\epsilon] = 0$ and $\mathbb{V}[\epsilon] = \tau^2$ (ϵ is not necessarily Gaussian)
- We construct an hypothesis for unknown f by fitting $\hat{f}_n$ to n training points

- $\hat{f}_n$ should mimic well f on unseen data aka should have low generalization error
- we also assume that the generalization error is the **expected squared error loss on an unseen example.**

- $\hat{f}_n$ contains **randomness** since it depends also on the errors ϵ_i
- (x_*, y_*) is an **unseen testing pair**

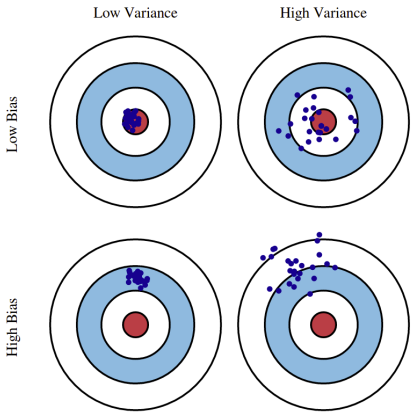
Then we can decompose its error:

$$\begin{aligned}\mathrm{MSE}\left(\hat{f}_n\right) &= \mathbb{E}_\epsilon\left[\left(y_* - \hat{f}_n(x_*)\right)^2\right] \\ &= \mathbb{E}\left[\left(\underbrace{\epsilon + f(x_*)}_{y_*} - \hat{f}_n(x_*)\right)^2\right]\end{aligned}$$

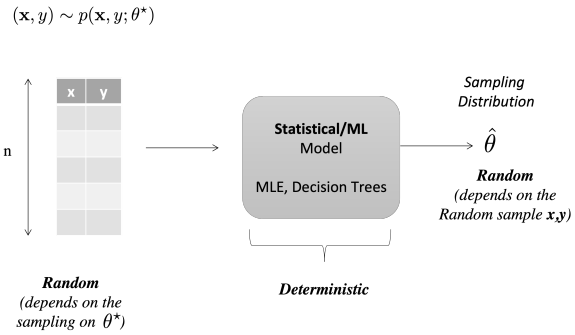
$$
\begin{aligned}
&= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[\left(f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right)^2\right] + \mathbb{E}\left[2\epsilon\left(f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right)\right] \\
&= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[\left(f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right)^2\right] + \underbrace{\mathbb{E}[\epsilon]}_{=0}\mathbb{E}\left[2\left(f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right)\right] \\
&= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[\left(f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right)^2\right] \\
&= \mathbb{E}\left[\epsilon^2\right] + \mathbb{E}\left[f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right]^2 + \mathbb{V}\left[f\left(x_*\right) - \hat{f}_n\left(x_*\right)\right] \quad \left(\mathbb{E}\left[X^2\right] = \mathbb{V}[X] + \mathbb{E}[X]^2\right) \\
&= \underbrace{\tau^2}_{\text{Irreducible error}} + \underbrace{\mathbb{E}\left(\left[\hat{f}_n\left(x_*\right) - f\left(x_*\right)\right]\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{V}\left[\hat{f}_n\left(x_*\right)\right]}_{\text{Variance}} \quad \left(\mathbb{V}[a - X] = \mathbb{V}[X]\right)
\end{aligned}
$$

Bias-Variance Tradeoff as Dartboard

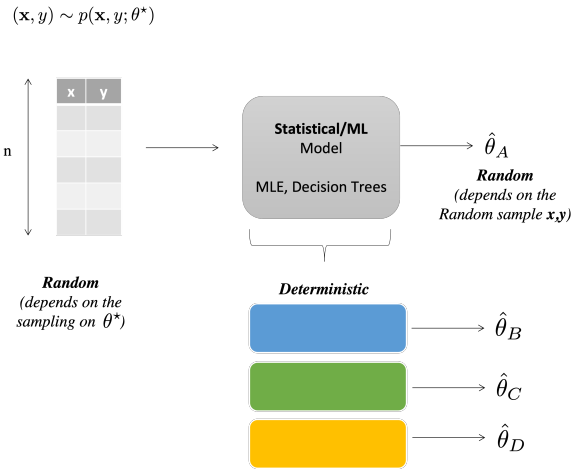
- Each dart shot below is a training experiment!



Sampling Distribution

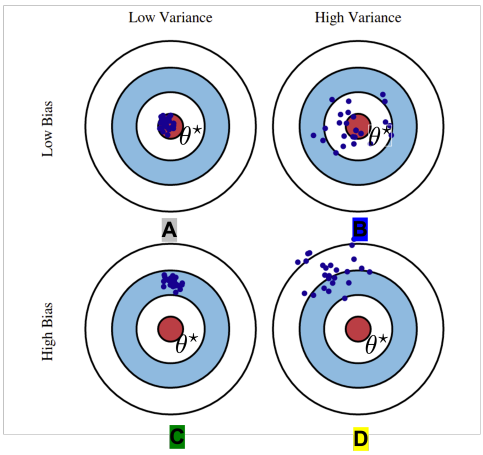


Sampling Distribution



Bias-Variance Tradeoff as Dartboard

- Each dart shot below is a training experiment!



Introduction to Supervised Learning

Assume that there is a unknown and complex generator $\mathcal{D}$ that provides output pairs $(\mathbf{x}, y)$.

- We refer to this **unknown generator process as an unknown probability distribution** $\mathcal{D}$ over input pairs $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ aka $\mathcal{D} \doteq p(\mathbf{x}, y)$.
- **Example:** Pairs of images and a label as in the case of bird/non-bird
  - $\mathbf{x}$ corresponds to the image;
  - y to the label

In practice, in a real-world problem **no one has access to** $\mathcal{D}$ **because problems are too complex**

> Try to write a computer program to generate all possible natural images that you can find in the word. Is it easy?

Let's assume here that we have access to $\mathcal{D}$ as a python function `get_prob_under_D(x,y)` that takes as input a pair `(x,y)` and returns the probability of the pair under $\mathcal{D}$.

If so, we can define the **Bayes optimal classifier** as the classifier that:

- for any test input $\mathbf{x}'$, simply returns the y' that maximizes `get_prob_under_D(x,y)`
- Or else, try all possible labels and return the label which yields maximum prob.

$$h(\mathbf{x}') = \arg\max_{y' \in \mathcal{Y}} p(\mathbf{x}', y') \tag{1}$$

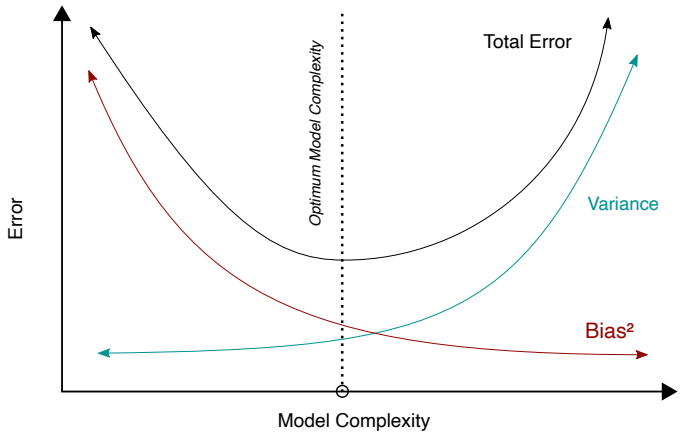
Bayes optimal classifier and Bias-Variance Tradeoff

$$\text{MSE}\left(\hat{f}_n\right) = \underbrace{\tau^2}_{\text{Irreducible error}} + \underbrace{\mathbb{E}\left(\left[\hat{f}_n(x_*) - f(x_*)\right]\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{V}\left[\hat{f}_n(x_*)\right]}_{\text{Variance}} \quad \text{with Bayes Optimal Classifier}$$

Bias and Variance terms go to zero!

$$\text{MSE}\left(\hat{f}_n\right) = \underbrace{\tau^2}_{\text{Irreducible error}}$$

Error in function of model complexity



1) Model selection and Cross-Validation

Learning = a) Lower ⬇ the cost $\mathcal{L}$ in training **AND** b) ⬇ also in test

In order to accomplish a), let's assume we have a **loss or cost** function:

$$\mathcal{L}(\underbrace{\hat{y}}_{pred.}, \underbrace{y}_{gt}) \quad \text{where} \quad \hat{y} = h_\theta(\mathbf{x})$$

- The job of $\mathcal{L}$ is to tell us how **"bad" a system's prediction is in comparison to the truth.**
- In particular, if y is what is defined as **ground-truth value or label** and $\hat{y}$ is machine prediction.
- You can view $\mathcal{L}$ as a measure of the system error over the training set.

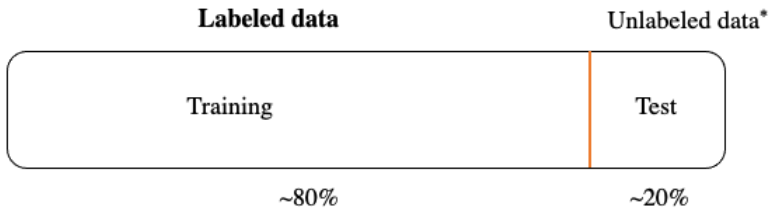
Goal of learning is do well on unseen samples (low generalization error)

1) Held Out Validation Set

A single validation set (or development set)

Ok we train on `train` split and we `test` on test split.

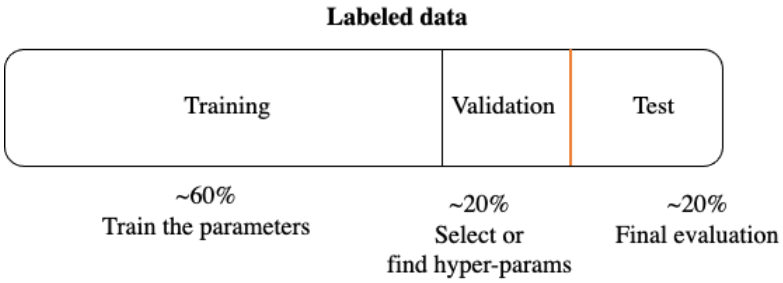
How do we select the distance and k neighbours?



*Most of the time you have label because you are not performing in reality
Beware of using it for evaluation performance because of overfit!

Ok we train on `train` split and we `test` on test split.

How do we select the distance and k neighbours?



Set size and partitioning: not a clear definition

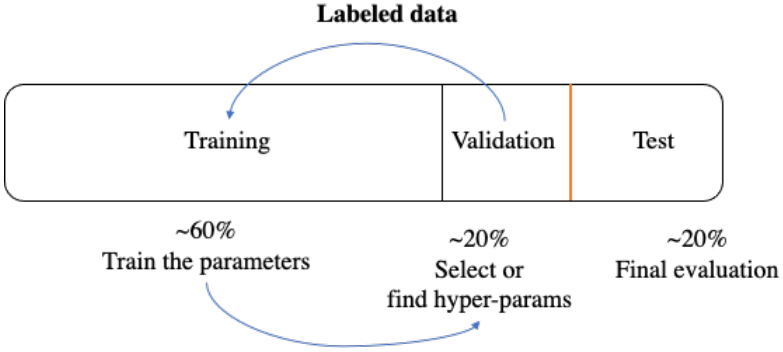
Train	70%	60%	80%
Validation *(dev)*	20%	20%	10%
Test	10%	20%	10%

Motivation for each split

- **[Training Set]** Do well on training-set by minimize the loss **(very huge optimistic perf.)**
- **[Validation Set]** Do well in generalization error **(moderate optimistic perf.)**
- **[Test Set]** Get an estimate of the generalization error **(no bias on perf.)**

Problem of a single held out split

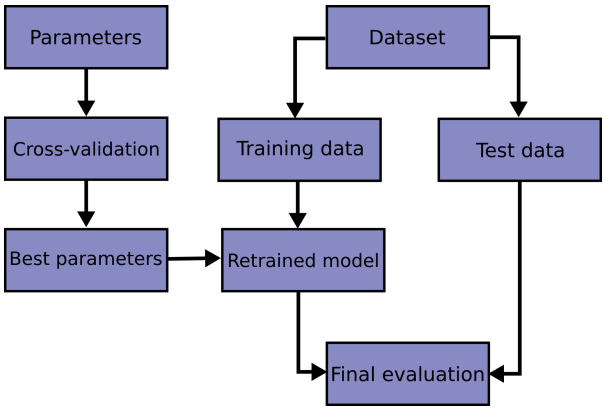
- It is **biased** to the **choice made by selecting** that specific validation partition.
- It **wastes part of the training data**
- Even though the hyper-params selected on the validation may be a few, there could be a chance of overfitting
- Think of each cycle "tune training/valid on validation", the dataset will **get rotten iteration over iteration**



Nested Cross-Validation

Nested Cross-Validation

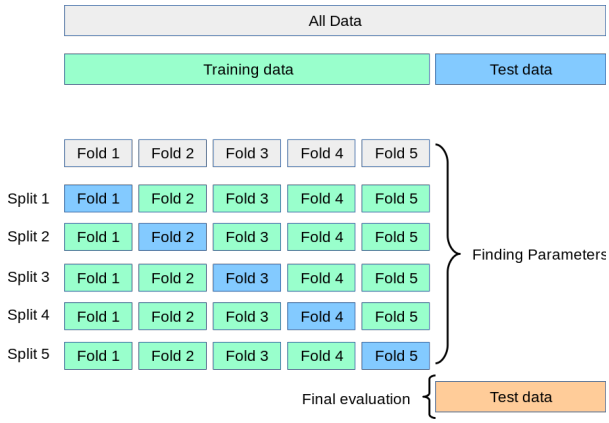
However, by partitioning the **available data into three sets, we drastically reduce the number of samples** which can be used for learning the model, and the **results can depend on a particular random choice** for the pair of (train, validation) sets.



Nested Cross-Validation

A solution to this problem is a procedure called **cross-validation (CV for short)**.

- A test set should still be held out for final evaluation,
- Validation set is no longer needed when doing CV.
- **K-fold CV**, the training set is **split into k smaller sets** (below K=5).



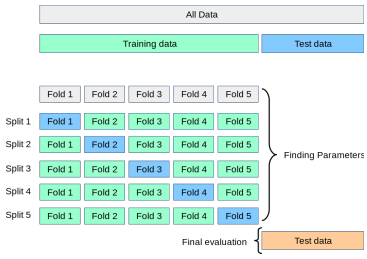
K-fold Cross-Validation

Pro:

- Reduces waste of data
- Unbiased with respect to the split choice
- Provides an estimate of standard deviation of your prediction (variance)

Con:

- **Computationally expensive!** *(though with multi-core you can run in parallel)*



Which value for K? (10)

In general:

- For larger K , the **error estimation tends to be more accurate**
- but computation time will be greater

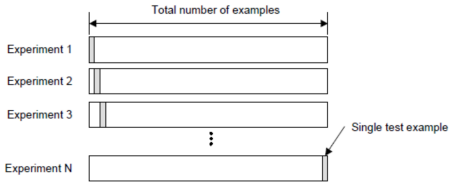
Rule of thumb:

- Typical choices for `K` are `2, 5, 10`.
- By far the most common is **K = 10: 10-fold cross validation.**
- Sometimes **5** is used for efficiency reasons.
- And sometimes **2 is used for subtle statistical reasons, but that is quite rare**

What if we have so little data that even K-fold won't work?

Leave-One-Out (LOO) Cross-Validation

- When $K = N$ then k-fold CV becomes **Leave-one-out (LOO) Cross-Validation** where N is the number of samples you have.
- **PRO**: may be applied in a low data regime
- **CONS**: it tends to overestimate generalization error



Leave-One-Out (LOO) Cross-Validation

- LOO is **more computationally expensive** than K-fold cross validation (not so expensive for *K-NN*).
- In terms of accuracy, **LOO often results in high variance as an estimator for the test error**.
- Since of the samples are used to build each model, models constructed from folds are virtually identical to each other and to the model built from the entire training set.

As a general rule, most authors, and empirical evidence, suggest that 5- or 10- fold cross validation should be preferred to LOO.

Cross-Validation Pseudo-Code

Note sklearn provides interface for this but is mandatory to have in your mind the entire flow:

Algorithm 8 CrossValidate(*LearningAlgorithm*, *Data*, K)

```
ê ← ∞                                          // store lowest error encountered so far
α̂ ← unknown                                    // store the hyperparameter setting that yielded it
for all hyperparameter settings α do
  err ← [ ]                                    // keep track of the K-many error estimates
  for k = 1 to K do
    train ← {(x_n, y_n) ∈ Data : n  mod K ≠ k − 1}
    test ← {(x_n, y_n) ∈ Data : n  mod K = k − 1}   // test every Kth example
    model ← Run LearningAlgorithm on train
    err ← err ⊕ error of model on test         // add current error to list of errors
  end for
  avgErr ← mean of set err
  if avgErr < ê then
    ê ← avgErr                                 // remember these settings
    α̂ ← α                                      // because they're the best so far
  end if
end for
```

CV works if the data is sampled i.i.d. from same distribution

- **[person identification]** We might try to classify every pixel in an image based on whether it contains a person or not.
- If we have 100 training images, each with 10,000 pixels, then we have a total of 1M training examples. The classification for a pixel in image 5 is highly dependent on the classification for a neighboring pixel in the same image. So if one of those pixels happens to fall in training data, and the other in development (or cross validation) data, your model will do unreasonably well.

- In this case, it is important that when you cross validate (or use development data), **you do so over images, not over pixels.**

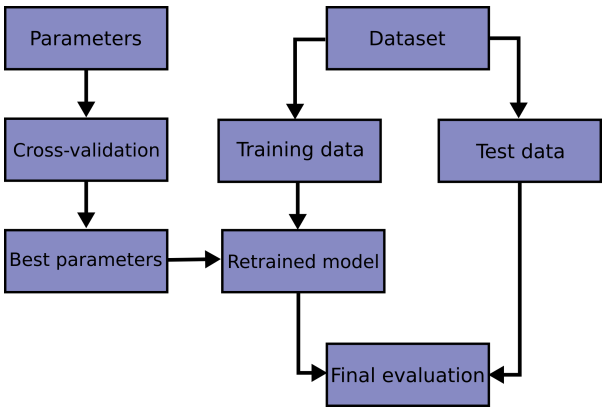
- The same goes for **text problems** where you sometimes want to classify things at a word level, but are handed a collection of documents. The important thing to keep in mind is that it is the **images (or documents) that are drawn independently from your data distribution and not the pixels (or words), which are drawn dependently.**

What to do after CV?

After running cross validation, you have **2 choices.**

1. You can either select **one of the K trained models as your final model** to make predictions
2. You can train a **new model on all of the data, using the hyperparameters selected by cross-validation.**

2. is generally preferred to 1.



Practical Example

We are working in the medical sector and we are using decision tree for their interpretability power, but we have to decide the **depth of the tree.**

Hyper-param Combinations	Training error ± std	10-fold Cross validation error ± std	Final Choice
Tree depth = 1			
Tree depth = 2			
Tree depth = 3			✓
Tree depth = 4			
Tree depth = 5			

Exam look-alike question: how many model (decision trees here) you need to train to make the choice?

CV Application Caveat

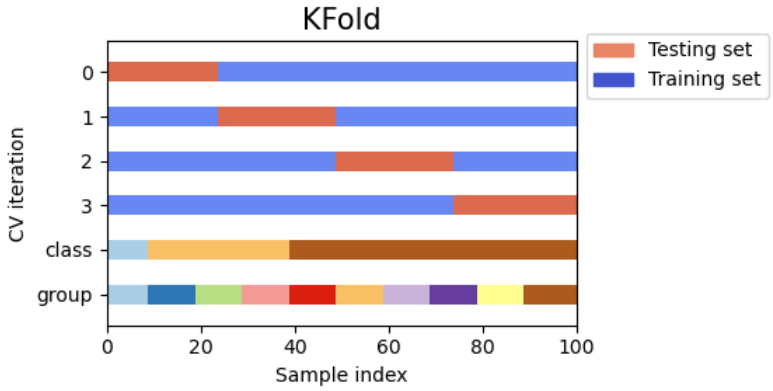
CV assumptions:

- Data is Independent and Identically Distributed (i.i.d.) aka, all samples stem from the same generative process
- The generative process is assumed to have no memory of past generated samples.

⚠️

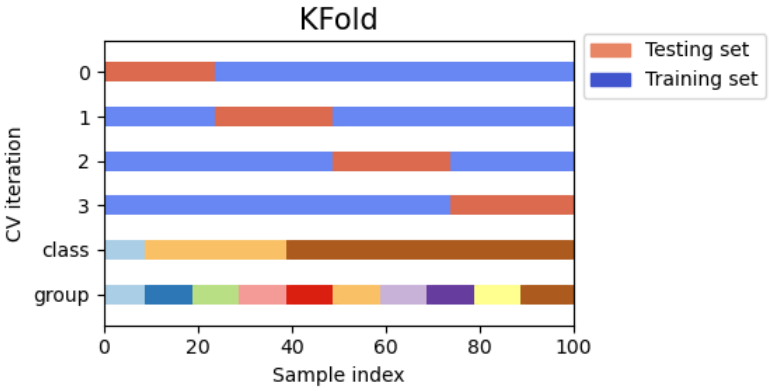
> **Note:** While i.i.d. data is a common assumption in machine learning theory, it rarely holds in practice. If one knows that the samples have been generated using a time-dependent process, it is safer to use a time-series aware cross-validation scheme. Similarly, if we know that the generative process has a group structure (samples collected from different subjects, experiments, measurement devices), it is safer to use group-wise cross-validation.

Truly i.i.d. $\rightarrow$ K-fold



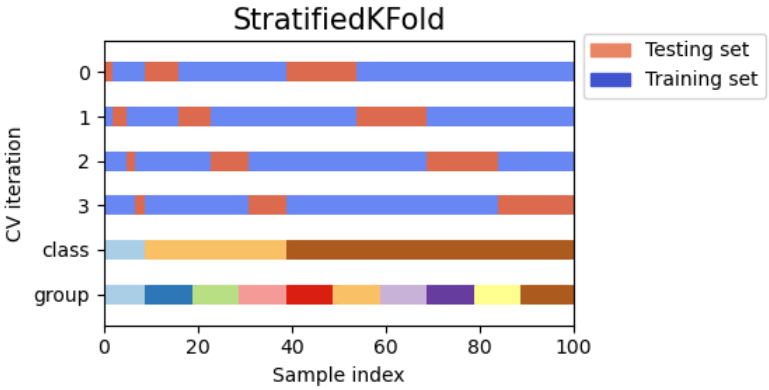
Truly i.i.d. → K-fold

- Unaware of class and groups



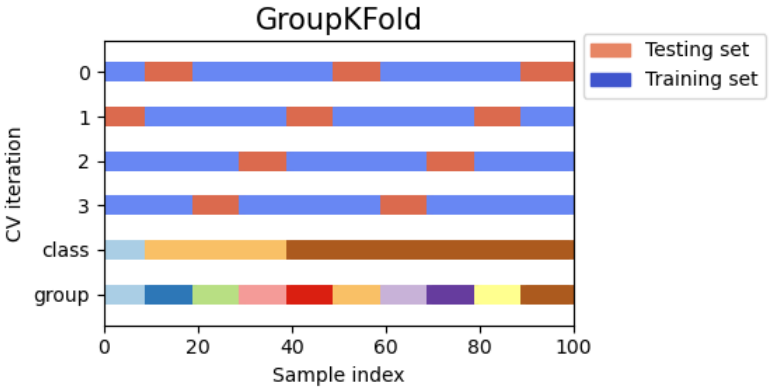
Strong Class Imbalance → Stratified K-fold

- Preserves the **class ratios in both train and test dataset.**



Unseen Group in test → Group k-fold

Ensures that the same group is **NOT** represented in both testing and training sets. *For example if the data is obtained from different subjects with several samples per-subject and if the model is flexible enough to learn from highly person specific features it could fail to generalize to new subjects.*



2) Hyper-parameter tuning of an estimator

Hyper-parameter: definition

Hyper-parameters are parameters that are not directly learnt within estimators.

- how should you set k in k-NN?
- how deep should the decision tree be?
- how many ensemble? (though you can use OOB)
- how many K centroids in K-means used a proxy for supervised learning
- how many layers of my Neural Networks?

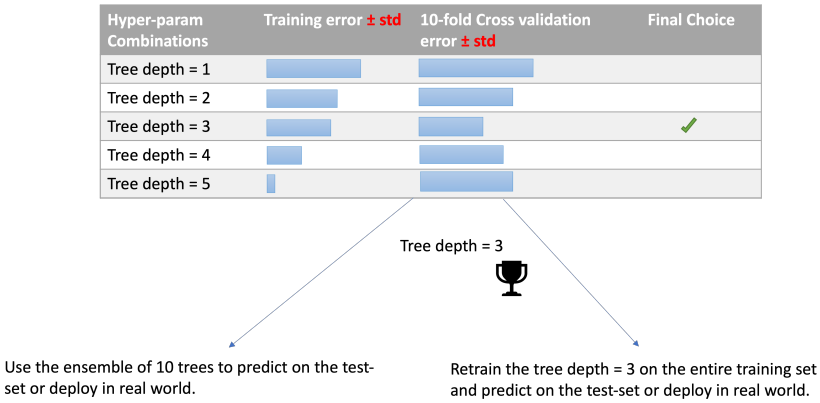
Recipe

A search of HP consists of:

- an estimator **(example: SVM or Decision Tree)**
- a parameter space; **($k \in [1, 3, 5, 7, 11]$)**
- a method for searching or sampling candidates; **(exhaustive grid search)**
- a cross-validation scheme; **(10-fold CV)**
- a score function **(accuracy)**

Hyper-parameter tuning

We are working in the medical sector and we are using decision tree for their interpretability power, but we have to decide the **depth of the tree.**



What happens if we have to chose/validate 2 hyper-params?

```
max_depth=None,  # we know the best for this is 3 but what about the other?
min_impurity_decrease=0.0
```

Grid Search on a "plane" of hyper-params

```
max_depth=None,  # we know the best for this is 3 but what about the other?
min_impurity_decrease=0.0
```

DPT = Tree Depth {1, 2, 3}
MID = Min impurity Decrease {0.01, 0.1}

Hyper-param Combinations	Training error ± std	10-fold Cross validation error ± std	Final Choice
DPT= 1, MID=0.01			
DPT= 1, MID=0.1			
DPT= 2, MID=0.01			✓
DPT= 2, MID=0.1			
DPT= 3, MID=0.01			
DPT= 3, MID=0.1			

Tree depth = 2, Min impurity Decrease = 0.01 

How many model do we train with k=10 fold cross-validation and grid search over depth $\in [1, 2, 3]$ and min impurity decrease in $\{0.01, 0.1\}$?

😱Grid Search on a "4D-cube" of hyper-params

```
max_depth=None,
min_impurity_decrease=0.0
min_samples_leaf=1,
max_leaf_nodes=None,
```

Method for searching

- **Exhaustive Grid Search** (brute force all the HP in CV given predefined ranges) **Mostly used**
- **Randomized Parameter Optimization** (distribution over parameters → sample; sample size or budget)
- **Searching for optimal parameters with successive halving** (tournament among candidates parameter)

Grid Search

```
# Set the parameters by cross-validation
tuned_parameters = [
    {"max_depth": [1, 2, 3, None],
     "min_impurity_decrease": [0.001, 0.01, 0.1, ],
     "criterion": ["gini", "entropy"],
    }
]
....
clf = GridSearchCV(DTC(), tuned_parameters, scoring="%s" % score)
```

In [3]:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier as DTC

# Loading the Digits dataset
digits = datasets.load_digits()

# To apply an classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
y = digits.target

# Split the dataset in two parts
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)

# Set the parameters by cross-validation
tuned_parameters = [
    {"max_depth": [1, 2, 3, None],
     "min_impurity_decrease": [0.001, 0.01, 0.1, ],
     "criterion": ["gini", "entropy"],
    }
]

scores = ["accuracy"]

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(DTC(), tuned_parameters, scoring="%s" % score)
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_["mean_test_score"]
    stds = clf.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds, clf.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()

# Note the problem is too easy: the hyperparameter plateau is too flat and the
# output model is the same for precision and recall with ties in quality.
```

```
# Tuning hyper-parameters for accuracy

Best parameters set found on development set:

{'criterion': 'entropy', 'max_depth': None, 'min_impurity_decrease': 0.001}

Grid scores on development set:

0.209 (+/-0.007) for {'criterion': 'gini', 'max_depth': 1, 'min_impurity_decrease': 0.001}
0.209 (+/-0.007) for {'criterion': 'gini', 'max_depth': 1, 'min_impurity_decrease': 0.01}
0.107 (+/-0.003) for {'criterion': 'gini', 'max_depth': 1, 'min_impurity_decrease': 0.1}
0.328 (+/-0.007) for {'criterion': 'gini', 'max_depth': 2, 'min_impurity_decrease': 0.001}
0.328 (+/-0.007) for {'criterion': 'gini', 'max_depth': 2, 'min_impurity_decrease': 0.01}
0.107 (+/-0.003) for {'criterion': 'gini', 'max_depth': 2, 'min_impurity_decrease': 0.1}
0.479 (+/-0.026) for {'criterion': 'gini', 'max_depth': 3, 'min_impurity_decrease': 0.001}
0.478 (+/-0.024) for {'criterion': 'gini', 'max_depth': 3, 'min_impurity_decrease': 0.01}
0.107 (+/-0.003) for {'criterion': 'gini', 'max_depth': 3, 'min_impurity_decrease': 0.1}
0.848 (+/-0.048) for {'criterion': 'gini', 'max_depth': None, 'min_impurity_decrease': 0.001}
0.786 (+/-0.071) for {'criterion': 'gini', 'max_depth': None, 'min_impurity_decrease': 0.01}
0.107 (+/-0.003) for {'criterion': 'gini', 'max_depth': None, 'min_impurity_decrease': 0.1}
0.200 (+/-0.013) for {'criterion': 'entropy', 'max_depth': 1, 'min_impurity_decrease': 0.001}
0.200 (+/-0.013) for {'criterion': 'entropy', 'max_depth': 1, 'min_impurity_decrease': 0.01}
0.200 (+/-0.013) for {'criterion': 'entropy', 'max_depth': 1, 'min_impurity_decrease': 0.1}
0.351 (+/-0.025) for {'criterion': 'entropy', 'max_depth': 2, 'min_impurity_decrease': 0.001}
0.351 (+/-0.025) for {'criterion': 'entropy', 'max_depth': 2, 'min_impurity_decrease': 0.01}
0.351 (+/-0.025) for {'criterion': 'entropy', 'max_depth': 2, 'min_impurity_decrease': 0.1}
0.527 (+/-0.039) for {'criterion': 'entropy', 'max_depth': 3, 'min_impurity_decrease': 0.001}
0.527 (+/-0.039) for {'criterion': 'entropy', 'max_depth': 3, 'min_impurity_decrease': 0.01}
0.519 (+/-0.050) for {'criterion': 'entropy', 'max_depth': 3, 'min_impurity_decrease': 0.1}
0.849 (+/-0.016) for {'criterion': 'entropy', 'max_depth': None, 'min_impurity_decrease': 0.001}
0.825 (+/-0.039) for {'criterion': 'entropy', 'max_depth': None, 'min_impurity_decrease': 0.01}
0.614 (+/-0.057) for {'criterion': 'entropy', 'max_depth': None, 'min_impurity_decrease': 0.1}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.92      0.89      0.91        27
           1       0.85      0.83      0.84        35
           2       0.96      0.72      0.83        36
           3       0.61      0.93      0.74        29
           4       0.89      0.83      0.86        30
           5       0.89      0.85      0.87        40
           6       0.95      0.89      0.92        44
           7       0.92      0.90      0.91        39
           8       0.79      0.95      0.86        39
           9       0.86      0.78      0.82        41

    accuracy                           0.86       360
   macro avg       0.87      0.86      0.86       360
weighted avg       0.87      0.86      0.86       360
```

Randomized Parameter Optimization

- A budget can be chosen independent of the number of parameters and possible values.
- Adding parameters that do not influence the performance does not decrease efficiency.

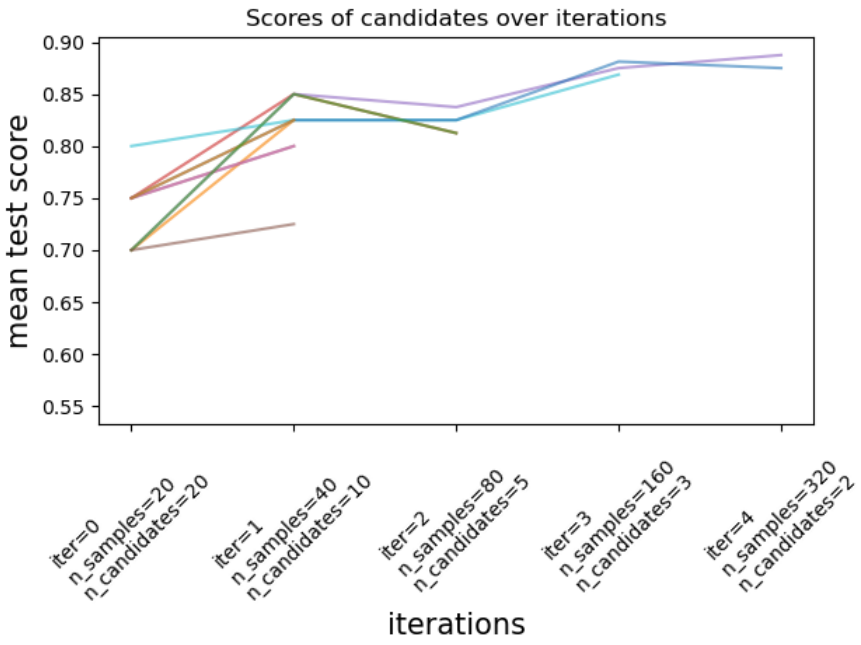
```
{'C': scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(scale=.1),
  'kernel': ['rbf'], 'class_weight':['balanced', None]}
```

```
n_iter int, default=10
Number of parameter settings that are sampled. n_iter trades off runtime vs quality of the
solution.
```

Searching for optimal parameters with successive halving

Idea: Successive halving (SH) is like a **tournament among candidate parameter combinations.**

- SH is an iterative selection process where all candidates (the parameter combinations) are evaluated with a small amount of resources at the first iteration.
- Only some of these candidates are selected for the next iteration, which will be allocated more resources.
  - For parameter tuning, the resource is typically the `number of training samples`, but it can also be an arbitrary numeric parameter such as `n_estimators` in a random forest.



Searching for optimal parameters with successive halving

Artificial Intelligence and Machine Learning

Unit II

Evaluation Metrics

OPIS for Course Evaluation

OPIS CODE: XJM26N6X

The code and guide are in the google classroom!

Guide on how to evaluate with OPIS (Opinion of the students):

https://www.uniroma1.it/sites/default/files/field_file_allegati/vademecum_per_studenti_opis_2022_23.pdf 🇮🇹

https://www.uniroma1.it/sites/default/files/field_file_allegati/guided_path_to_access_student_s_opinions_questionnaire_2022_2023.pdf 🇺🇸

Before moving to metrics, let's see hypothetical question about Decision trees in the exam

Decision Tree Sample Question in the Exam

Person ID (training example)	Overcooked pasta?	Waiting Time	Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

1. Using the training data, construct **a decision tree** for the **binary classification** of customers of the restaurant "Mama's Pasta" into 'Satisfied (1)' or 'Unsatisfied (0)'.
2. Use the **Information Gain (IG)** as the decision criterion to select which attribute to split on. Show your calculations for the IG for all possible attributes for every split.

Taken from U. of Toronto; you can practice more with this

Decision Tree Sample Question in the Exam

Person ID (training example)	[Feat. 1] Overcooked pasta?	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

How to approach this, First, Analysis:

- the features $\mathbf{x}$ are not in $\mathbb{R}$ but **categorical variables** in 3 dimensions. In some sense, it is easier to do it manually with variables like this.
  - Feature 1 is $\{Yes, No\}$
  - Feature 2 is $\{Long, Short\}$
  - Feature 3 is $\{Yes, No\}$
- the target y is **binary categorical variable** $\{Satisfied\ (yes), UnSatisfied\ (no)\}$

Sketch of Solution

Person ID (training example)	[Feat. 1] Overcooked pasta?	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

- What is the Entropy of the y with **no split taken**?
  - How many training samples we have: *5*
  - **Over 5 samples** we have **3 yes and 2 no**

So the starting set if $|S| = 5$, and $H(S) = -\sum_{y \in \{yes, no\}} p(y) \log_2(p(y))$

Over S, $p(y == yes) = \frac{3}{5}$ while $p(y == no) = \frac{2}{5}$

So, $H(S) = -\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5}) \approx 0.971$

Sketch of Solution

Person ID (training example)	[Feat. 1] Overcooked pasta?	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

- $H(S) = -\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5}) \approx 0.971$
- Now we have to just "brute force" all the possible splits (3) as if we took the split and compute the weighted entropy after.
- Split on **Overcooked pasta** and compute the new Entropy. I have two case:
  - A) $\rightarrow$ `Ov. pasta==Yes` I have 3 samples over the "parent" 5 so i have to weight this as $\frac{3}{5}$
  - B) `Ov. pasta==No` I have 2 samples over the "parent" 5 so i have to weight this as $\frac{2}{5}$

$\frac{3}{5}H(y \mid \texttt{Ov. pasta==Yes}) = \ \texttt{?}$

Sketch of Solution

Person ID (training example)	[Feat. 1] Overcooked pasta?	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

- $H(S) = -\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5}) \approx 0.971$
- Now we have to just "brute force" all the possible splits (3) as if we took the split and compute the weighted entropy after.
- Split on **Overcooked pasta** and compute the new Entropy. I have two case:
  - A) `Ov. pasta==Yes` I have 3 samples over the "parent" 5 so i have to weight this as $\frac{3}{5}$
  - B) `Ov. pasta==No` I have 2 samples over the "parent" 5 so i have to weight this as $\frac{2}{5}$
- Now **look at the labels** for `Ov. pasta==Yes`

$$\frac{3}{5}H(y \mid \texttt{Ov. pasta==Yes}) = \frac{3}{5} \cdot \left[- \sum_{y \in \{yes, no\}} p(y|x) \log_2(p(y|x)) \right]$$

we have **2 No Satisfaction and 1 Yes,Satisfaction , over 3 samples after choosing**:

$$\frac{3}{5}H(y \mid \texttt{Ov. pasta==Yes}) = \frac{3}{5} \cdot [\underbrace{-\frac{1}{3}\log_2(\frac{1}{3})}_{yes} - \underbrace{\frac{2}{3}\log_2(\frac{2}{3})}_{no}] \approx 0.551$$

Sketch of Solution

Person ID (training example)	[Feat. 1] Overcooked pasta?	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

- $H(S) = -\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5}) \approx 0.971$
- Now we have to just "brute force" all the possible splits (3) as if we took the split and compute the weighted entropy after.
- Split on **Overcooked pasta** and compute the new Entropy. I have two case:
  - A) `Ov. pasta==Yes` I have 3 samples over the "parent" 5 so i have to weight this as $\frac{3}{5}$
  - B) $\rightarrow$ `Ov. pasta==No` I have 2 samples over the "parent" 5 so i have to weight this as $\frac{2}{5}$
- Now **look at the labels** for `Ov. pasta==No`

$$\frac{2}{5}H(y \mid \texttt{Ov. pasta==No}) = \frac{2}{5} \cdot \left[- \sum_{y \in \{yes, no\}} p(y|x) \log_2(p(y|x)) \right]$$

we have **0 UnSatisfaction and 2 Yes, Satisfaction , over 2 samples after choosing**:

$$\frac{2}{5}H(y \mid \texttt{Ov. pasta==No}) = \frac{2}{5} \cdot \big[- \frac{0}{2}\log_2(\frac{0}{2}) - \frac{2}{2}\log_2(\frac{2}{2}) \big] = 0$$

Sketch of Solution

Person ID (training example)	[Feat. 1] Overcooked pasta?	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	Yes	Long	No	1 (yes)
$\mathbf{x}_2$	No	Short	Yes	1 (yes)
$\mathbf{x}_3$	Yes	Long	Yes	0 (no)
$\mathbf{x}_4$	No	Long	Yes	1 (yes)
$\mathbf{x}_5$	Yes	Short	Yes	0 (no)

- $H(S) = -\frac{3}{5}\log_2(\frac{3}{5}) - \frac{2}{5}\log_2(\frac{2}{5}) \approx 0.971$
- Now we have to just "brute force" all the possible splits (3) as if we took the split and compute the weighted entropy after.
- Split on **Overcooked pasta** and compute the new Entropy. I have two case:
  - A) `Ov. pasta==Yes` I have 3 samples over the "parent" 5 so i have to weight this as $\frac{3}{5}$
  - B) `Ov. pasta==No` I have 2 samples over the "parent" 5 so i have to weight this as $\frac{2}{5}$

Information Gain for split on `Overcooked pasta`:

$$IG(Y \mid \texttt{Ov. pasta}) \doteq \frac{5}{5}H(S) - \left[\frac{2}{5}H(y \mid \texttt{Ov. pasta==No}) + \frac{3}{5}H(y \mid \texttt{Ov. pasta==Yes}) \right]$$

$$IG(Y \mid \texttt{Ov. pasta}) \doteq 0.971 - [0.551 + 0] \approx 0.42$$

We are done only for 1/3 of the features.

Now we have to keep repeating this for the other 2 features and then split on the feature with the maximum IG </ins>

Split Feature	IG
Overcooked Pasta	0.42
Waiting Time	?
Rude Waiter	?

Fast Forward

Split Feature	IG
Overcooked Pasta	**0.42**
Waiting Time	0.020
Rude Waiter	0.171

So we split on `Overcooked Pasta over Yes` with a gain of 0.42

- Note that `Overcooked Pasta` is a binary variable so splitting over Yes or Not does not matter.
- If `Overcooked Pasta` were $\{Yes, No, Maybe\}$ we should have splitted taking note of the threshold. -i.e. if we binary split on **Yes**, the other split is $\neg$ `Yes` (which means it could be `No` or `Maybe`).

So now the problem is:

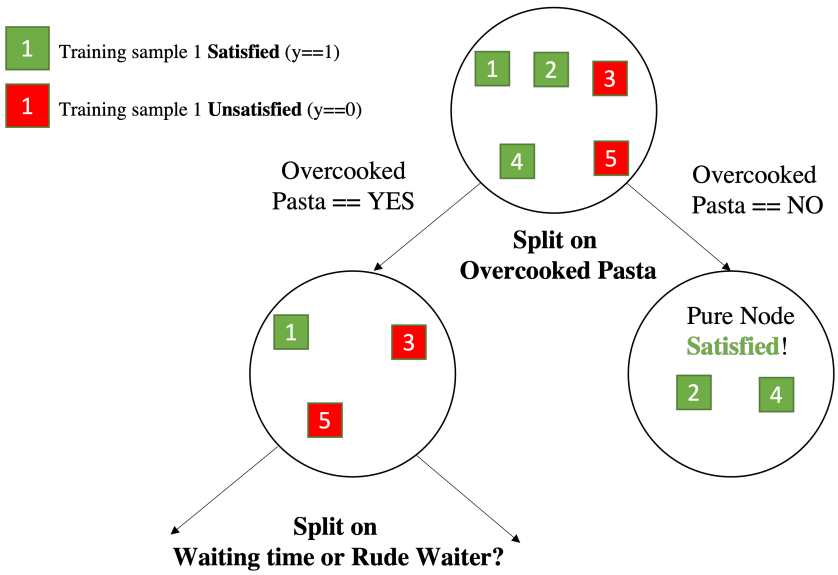
Split Overcooked pasta == Yes

Person ID (training example)	~~[Feat. 1] Overcooked pasta?~~	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_1$	~~Yes~~	Long	No	1 (yes)
$\mathbf{x}_3$	~~Yes~~	Long	Yes	0 (no)
$\mathbf{x}_5$	~~Yes~~	Short	Yes	0 (no)

Split Overcooked pasta == No

Person ID (training example)	~~[Feat. 1] Overcooked pasta?~~	[Feat. 2] Waiting Time	[Feat. 3] Rude Waiter?	Satisfied y
$\mathbf{x}_2$	~~No~~	Short	Yes	1 (yes)
$\mathbf{x}_4$	~~No~~	Long	Yes	1 (yes)

And the tree is:



3) Metrics for evaluation [mainly for classification]

Why evaluation metrics

- Training objective (cost function) is only a proxy for real world objective.
- Metrics help capture a business goal into a quantitative target (not all errors are equal).
- Helps organize ML team effort towards that target.
- Generally in the form of improving that metric on the dev set.
- Useful to quantify the "gap" between:
  - Desired performance and baseline (estimate effort initially).
  - Desired performance and current performance.
- Useful for lower level tasks and debugging (like diagnosing bias vs variance).
- Ideally training objective should be the metric, but not always possible. Still, metrics are useful and important for evaluation.

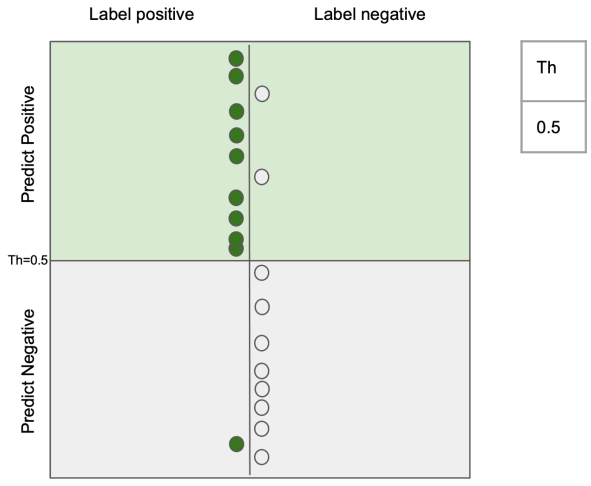
Binary Classification

- $\mathbf{x}$ is input, y is binary takes value in $\{0, 1\}$
- The model is $\hat{y} = h(\mathbf{x})$
- Two type of models:
  1. Models that outputs a categorical class directly (K-NN, Decision Trees)
  2. Model that outputs a **real value aka probability** $\in [0, 1]$ (Logistic Regression, SVM)

Score of Binary Classifier

Picture taken from Stanford Slide (copyright by Anand Avati)

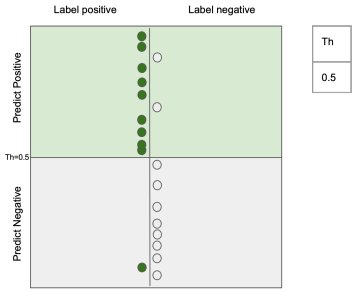
Score of Binary Classifier works with a Threshold



Picture taken from Stanford Slide

Easy Peasy as Accuracy

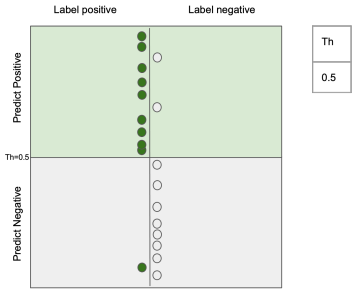
$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$



Picture taken from Stanford Slide

Relaxation of Accuracy: Top-K Accuracy

$$\texttt{top-k accuracy}(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \sum_{j=1}^{k} 1(\hat{f}_{i,j} = y_i)$$



Picture taken from Stanford Slide

Beyond Accuracies: Confusion Matrix

Beyond Accuracies: Confusion Matrix (Binary Case)

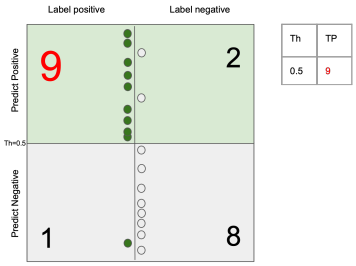
Picture taken from Stanford Slide

Confusion Matrix (Multi-class case)

```
from sklearn.metrics import confusion_matrix
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
confusion_matrix(y_true, y_pred)
> array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

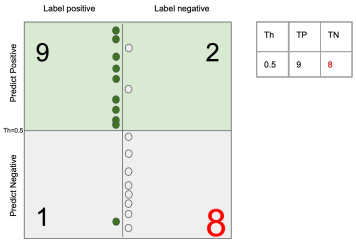
{{import matplotlib.pyplot as plt; from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay;y_true = [2, 0, 2, 2, 0, 1];y_pred = [0, 0, 2, 2, 0, 2];disp = ConfusionMatrixDisplay(confusion_matrix(y_true, y_pred));disp.plot();plt.show()}}

True Positive



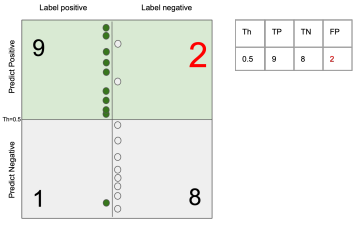
Picture taken from Stanford Slide

True Negative



Picture taken from Stanford Slide

False Positive



Picture taken from Stanford Slide

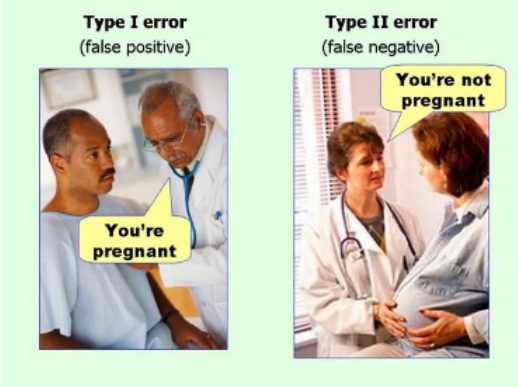
False Negative

Th	TP	TN	FP	FN
0.5	9	8	2	1

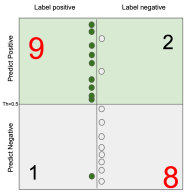
Picture taken from Stanford Slide

Next slide will make things more clear

FP and FN also called Type-1 and Type-2 errors



New view on Accuracy



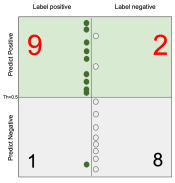
Th	TP	TN	FP	FN	Acc
0.5	9	8	2	1	.85

Picture taken from Stanford Slide

Precision

Ability to return only images that match the query

$$\text{precision} = \frac{tp}{tp + fp}$$



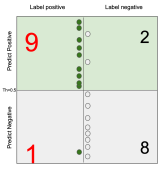
Th	TP	TN	FP	FN	Acc	Pr
0.5	9	8	2	1	.85	.81

Picture taken from Stanford Slide

Recall

Ability to return all the images that match the query

$$\text{recall} = \frac{tp}{tp + fn}$$



Th	TP	TN	FP	FN	Acc	Pr	Recall
0.5	9	8	2	1	.85	.81	.9

Picture taken from Stanford Slide

Precision, Recall, F-Measure

$$\text{precision} = \frac{tp}{tp + fp}$$

$$\text{recall} = \frac{tp}{tp + fn}$$

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}$$

True Positive Rate, False Positive Rate

$$\text{TPR} = \frac{tp}{P_{gt}}$$

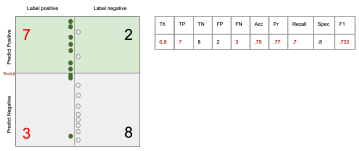
$$\text{FPR} = \frac{fp}{N_{gt}}$$

Summary

Sources: [5][6][7][8][9][10][11][12] view · talk · edit

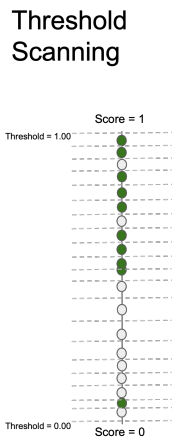
		Predicted condition			
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) = TPR + TNR − 1	Prevalence threshold (PT) = $\frac{\sqrt{\text{TPR} \times \text{FPR}} - \text{FPR}}{\text{TPR} - \text{FPR}}$
Actual condition	**Positive (P)**	**True positive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

Change Threshold → New Confusion Matrix



Picture taken from Stanford Slide

Threshold Scanning → ROC Curve

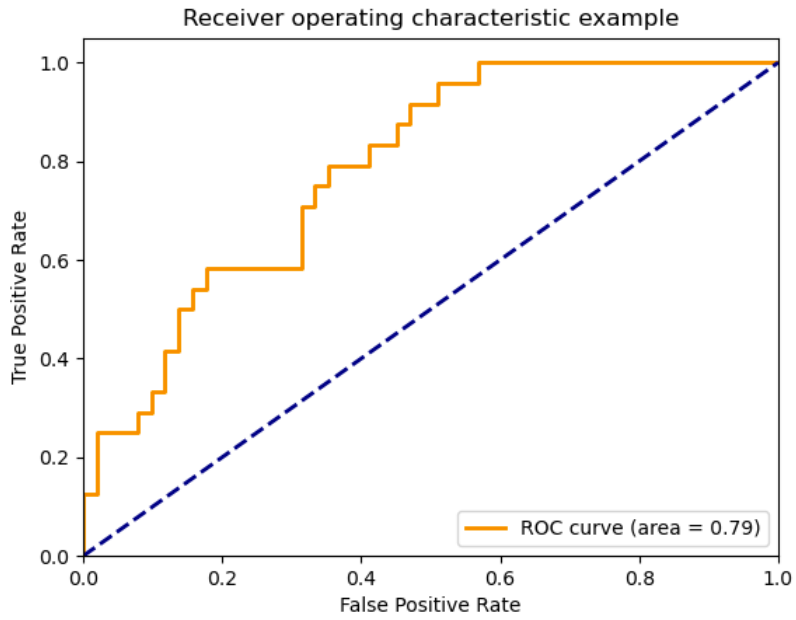


Threshold	TP	TN	FP	FN	Accuracy	Precision	Recall	Specificity	F1
1.00	0	10	0	10	0.50	1	0	1	0
0.95	1	10	0	9	0.55	1	0.1	1	0.182
0.90	2	10	0	8	0.60	1	0.2	1	0.333
0.85	2	9	1	8	0.55	0.667	0.2	0.9	0.308
0.80	3	9	1	7	0.60	0.750	0.3	0.9	0.429
0.75	4	9	1	6	0.65	0.800	0.4	0.9	0.533
0.70	5	9	1	5	0.70	0.833	0.5	0.9	0.625
0.65	5	8	2	5	0.65	0.714	0.5	0.8	0.588
0.60	6	8	2	4	0.70	0.750	0.6	0.8	0.667
0.55	7	8	2	3	0.75	0.778	0.7	0.8	0.737
0.50	8	8	2	2	0.80	0.800	0.8	0.8	0.800
0.45	9	8	2	1	0.85	0.818	0.9	0.8	0.857
0.40	9	7	3	1	0.80	0.750	0.9	0.7	0.818
0.35	9	6	4	1	0.75	0.692	0.9	0.6	0.783
0.30	9	5	5	1	0.70	0.643	0.9	0.5	0.750
0.25	9	4	6	1	0.65	0.600	0.9	0.4	0.720
0.20	9	3	7	1	0.60	0.562	0.9	0.3	0.692
0.15	9	2	8	1	0.55	0.529	0.9	0.2	0.667
0.10	9	1	9	1	0.50	0.500	0.9	0.1	0.643
0.05	10	1	9	0	0.55	0.526	1	0.1	0.690
0.00	10	0	10	0	0.50	0.500	1	0	0.667

Picture taken from Stanford Slide

ROC - Receiver Operating Characteristic curve

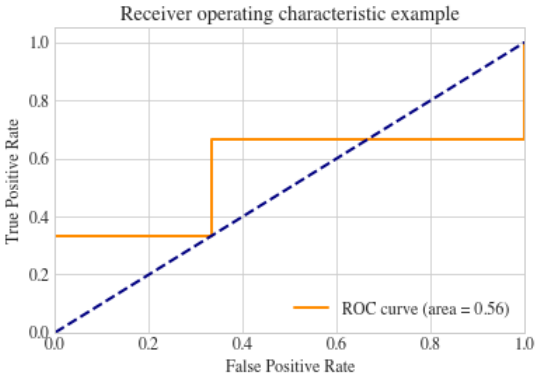
- ROC curve, is a **graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied.**
- It is created by plotting the **fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings.**
- TPR is also known as *sensitivity*, and FPR is one minus the *specificity or true negative rate*.
- ROC and Area under the ROC (AUC) **do not require to compute a threshold**



Computing ROC "manually"

In [4]:
```
y = np.array([0, 0, 1, 1, 1, 0])
scores = np.array([0.1, 0.4, 0.35, 0.8, 0.01, 0.2])
```

In [5]:
```
import numpy as np
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y, scores, drop_intermediate=False)
plt.figure()
lw = 2
plt.plot(
    fpr,
    tpr,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % auc(fpr, tpr),
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```



In [6]:
```
tpr, fpr, thresholds
```

Out[6]:
```
(array([0.        , 0.33333333, 0.33333333, 0.66666667, 0.66666667,
        0.66666667, 1.        ]),
 array([0.        , 0.        , 0.33333333, 0.33333333, 0.66666667,
        1.        , 1.        ]),
 array([1.8 , 0.8 , 0.4 , 0.35, 0.2 , 0.1 , 0.01]))
```

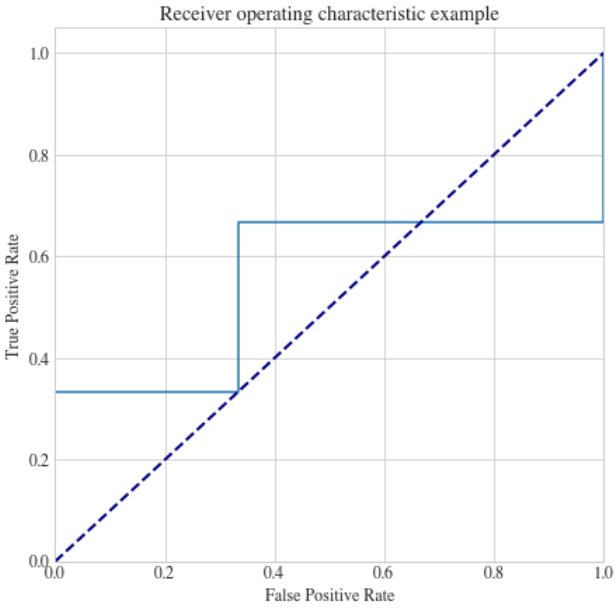
In [7]:
```
def thrs_count(scores, index, denom, thrs, normalize=True):
    count = sum(scores[index] >= thrs)
    return count/denom if normalize else count


pos = y == 1  # indexing the positive
neg = y == 0  # indexing the negative
n_pos = sum(pos)  # sum the ground-truth positve
n_neg = sum(neg)  # sum the ground-truth negative
sort_scores = sorted(scores, reverse=True)  # sort highest first
sort_scores = [sort_scores[0] + 1] + sort_scores  # add extra point
ROC = [(thrs_count(scores, pos, n_pos, thrs), thrs_count(scores, neg, n_neg, thrs))
       for thrs in sort_scores] # for all thresholds, count TP and FP and normalize them
my_TPR, my_FPR = list(map(lambda *args: args, *ROC)) # reshape
```

In [8]:
```
my_TPR, my_FPR, sort_scores
```

Out[8]:
```
((0.0,
  0.3333333333333333,
  0.3333333333333333,
  0.6666666666666666,
  0.6666666666666666,
  0.6666666666666666,
  1.0),
 (0.0,
  0.0,
  0.3333333333333333,
  0.3333333333333333,
  0.6666666666666666,
  1.0,
  1.0),
 [1.8, 0.8, 0.4, 0.35, 0.2, 0.1, 0.01])
```

In [9]:
```
assert all([np.allclose(tpr, my_TPR),
            np.allclose(fpr, my_FPR),
            np.allclose(thresholds, sort_scores)]), 'Your ROC is wrong'
```

In [10]:
```
plt.figure(figsize=(7,7))
plt.plot(my_FPR,my_TPR);
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.show()
```

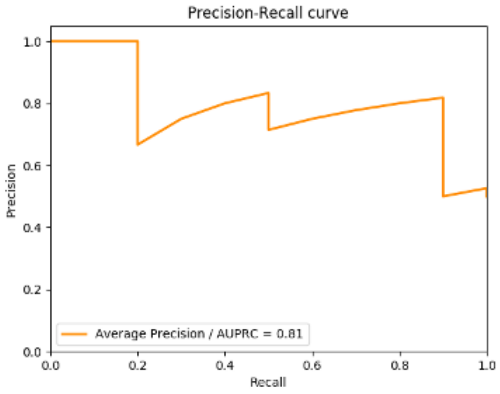


In [111]:
```
AUC = np.dot(np.diff(fpr), tpr[1:])
print(AUC)
assert np.allclose(AUC, auc(fpr, tpr)), 'my AUC is WRONG'
```

```
0.5555555555555556
```

Precision Recall Curve

- Same as ROC but plot **Precision in function of the Recall**
- Unlike ROC can have **zig-zag (non-monotonic) trends**
  - Getting a bunch of positives in the sequence increases both precision and recall (hence curve climbs up slowly)...
  - but getting a bunch of negatives in the sequence drops the precision without increasing recall. Hence, by definition, curve has only slow climbs and vertical drops
- Instead of AUC we compute **AP (Average Precision)**
  - *Intuition: By randomly picking the threshold, what's the expected precision?*
- Used in ranking systems (Image Retrieval)



Precision, Recall, F-Measure

$$\text{precision} = \frac{tp}{tp+fp}$$

$$\text{recall} = \frac{tp}{tp+fn}$$

$$F_\beta = (1+\beta^2)\frac{\text{precision}\times\text{recall}}{\beta^2\text{precision}+\text{recall}}$$

Class Imbalance: a big problem

Symptom: Prevalence < 5% (no strict definition) $\frac{P_{gt}}{P_{gt}+N_{gt}}$

- **Accuracy:** Blindly predict majority class you will be mostly right.
- **Log-Loss (penalize wrongly confident predictions):** Majority class can dominate the loss.

$$L_{\log}(y,p) = -\log \Pr(y|p) = -\big(y\log(p) + (1-y)\log(1-p)\big)$$

- **AUROC:** Easy to keep AUC high by scoring most negatives very low.
- **AUPRC:** Somewhat more robust than AUROC. But other challenges.

`In general:  Accuracy << AUROC << AUPRC`

Balanced Accuracy

Avoids inflated performance estimates on imbalanced datasets. It is the raw accuracy where each sample **is weighted according to the inverse prevalence of its true class.**

Thus for balanced datasets, the score is equal to accuracy.

In the binary case, balanced accuracy is equal to the arithmetic mean of sensitivity (true positive rate) and specificity (true negative rate):

$$\texttt{balanced-accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Homework (sample question in an exam)

You are given a machine learning system performing binary classification on 6 samples. The ground-truth labels and the unnormalized scores for the six samples are:

labels	**-1**	**1**	**-1**	**1**	**-1**	**1**
score	1000	0.5	-0.9	0.8	-0.1	0.1

The scores are unnormalized (i.e. not probabilities) and **correlated** with the positive label (1).

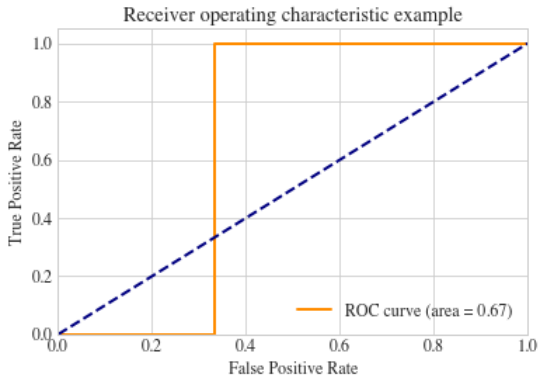
- For labels **1** the higher the score the better it is;
- whereas for labels **-1** the lower the score, the better it is.

1. Compute the ROC with a table and/or draw it approximatively the ROC curve (TPR vs FPR)
2. Calculate the Area Under the Curve (AUC).
3. How woud you set the score to make $AUC = 100\%?$ | **labels** | -1 | 1 | -1 | 1 | -1 | 1 | |--------|----|-----|------|-----|------|-----| | **score** | ? | ? | ? | ? | ? | ? |
4. How woud you set the score to make $AUC = 0\%?$ | **labels** | -1 | 1 | -1 | 1 | -1 | 1 | |--------|----|-----|------|-----|------|-----| | **score** | ? | ? | ? | ? | ? | ? |
5. Considering the score of answer 3, that gives $AUC = 100\%?$, what happens if you multiply the scores by a constant positive value $\alpha > 0$? Is the ROC going to change? What about the AUC?

Solution

In [12]:
```
y = np.array([-1, 1, -1, 1, -1, 1])
scores = np.array([1000, 0.5, -0.9, 0.8, -0.1, 0.1])
```

In [13]:
```python
import numpy as np
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y, scores, drop_intermediate=False, pos_label=1)
plt.figure()
lw = 2
plt.plot(
    fpr,
    tpr,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % auc(fpr, tpr),
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```



In [14]:
```python
pos = y == 1  # indexing the positive
neg = y == -1  # indexing the negative
n_pos = sum(pos)  # sum the ground-truth positve
n_neg = sum(neg)  # sum the ground-truth negative
sort_scores = sorted(scores, reverse=True)  # sort highest first
sort_scores = [sort_scores[0] + 1] + sort_scores  # add extra point
ROC = [(thrs_count(scores, pos, n_pos, thrs, normalize=False),
        thrs_count(scores, neg, n_neg, thrs, normalize=False))
       for thrs in sort_scores]  # for all thresholds, count TP and FP and normalize them
diff_fpr = np.diff(np.array(my_FPR))
my_TPR, my_FPR = list(map(lambda *args: args, *ROC))  # reshape
table = '|thrs  |tpr    |fpr    |diff_fpr       |\n|---         |---    |---    |---    \n'
for count, (mtpr, mfpr, thrs) in enumerate(zip(my_TPR, my_FPR, sort_scores)):
    diff_fpr = mfpr/n_neg-my_FPR[count-1]/n_neg
    table += f'|{thrs}  |{mtpr}/{n_pos}         |{mfpr}/{n_neg}         | { str(diff_fpr)[:5] if count > 0 else
print(table)
```

```
|thrs   |tpr    |fpr    |diff_fpr       |
|---    |---    |---    |---
|1001.0         |0/3    |0/3    | 0
|1000.0         |0/3    |1/3    | 0.333
|0.8    |1/3    |1/3    | 0.0
|0.5    |2/3    |1/3    | 0.0
|0.1    |3/3    |1/3    | 0.0
|-0.1   |3/3    |2/3    | 0.333
|-0.9   |3/3    |3/3    | 0.333
```

ROC Table

{{print(table)}}

AUC

$$\mathbf{AUC} = \sum_{t=1}^{T} \delta_{\mathrm{FPR}_t} \cdot \mathrm{TPR}_t = \boldsymbol{\delta}_{\mathrm{FPR}}^{T} \cdot \mathbf{TPR}$$

$$\mathbf{AUC} = \underbrace{\frac{1}{3} \cdot 0}_{t=1000} + \underbrace{0 \cdot \frac{1}{3}}_{t=0.8} + \underbrace{0 \cdot \frac{2}{3}}_{t=0.5} + \underbrace{0 \cdot \frac{3}{3}}_{t=0.1} + \underbrace{\frac{1}{3} \cdot \frac{3}{3}}_{t=-0.1} + \underbrace{\frac{1}{3} \cdot \frac{3}{3}}_{t=-0.9} = \frac{2}{3}$$

- $\delta_{\mathrm{FPR}_t} = FPR_t - FPR_{t-1}$

In [15]:
```
my_FPR/n_neg
```

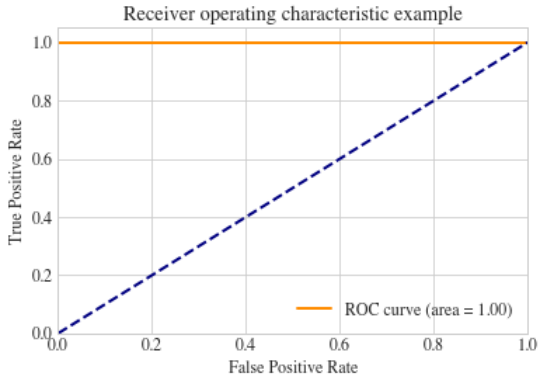
Out[15]:
```
array([0.        , 0.33333333, 0.33333333, 0.33333333, 0.33333333,
       0.66666667, 1.        ])
```

In [16]:
```
np.dot(np.diff(my_FPR/n_neg), (my_TPR/n_pos)[1:])
```

Out[16]:
```
0.6666666666666667
```

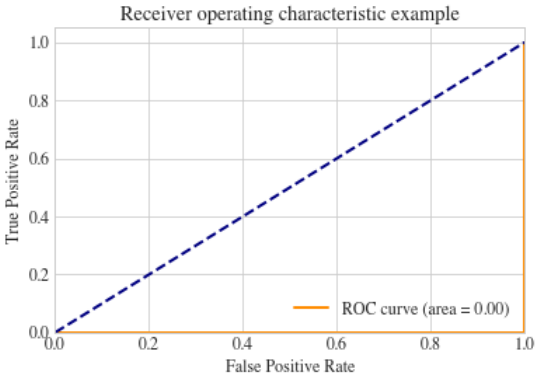
In [17]:
```
y = np.array([-1, 1, -1, 1, -1, 1])
alpha = 1
scores = np.array([-alpha, alpha, -alpha, alpha, -alpha, alpha])
```

In [18]:
```
import numpy as np
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y, scores, drop_intermediate=False, pos_label=1)
plt.figure()
lw = 2
plt.plot(
    fpr,
    tpr,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % auc(fpr, tpr),
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```



In [19]:
```
y = np.array([-1, 1, -1, 1, -1, 1])
alpha = -1
scores = np.array([-alpha, alpha, -alpha, alpha, -alpha, alpha])
```

In [20]:
```
import numpy as np
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y, scores, drop_intermediate=False, pos_label=1)
plt.figure()
lw = 2
plt.plot(
    fpr,
    tpr,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % auc(fpr, tpr),
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 0.00)
0.0
0.2
0.4
0.6
0.8
1.0