Model requirements

| **Visualization** | **Ensembling method** | **Local model** |  |
| --- | --- | --- | --- |
| **simple prediction -> provide just a label** | Any -> best performance possible  (hard to explain -> black box) | Any classifier |  |
| **Rich visualization (features importance, white box model, prediction probability)** | Stacking with :   * decision tree * … * (easy to understand) | “informative learners” :   * decision tree * … |  |
| **Probability distribution of the class** | Combine the distributions of each local model | Share the distributions  (just mean and std? Supposing the distributions are gaussian? ) -> what distribution should we share ? AFclass on which features? |  |
|  |  |  |  |

Ensemble methods :

* Bagging (black-box approach, works with complex models -> helps to reduce the variance)
* Voting:
  + Hard
  + Soft
* Averaging ( weighted or not )
* Boosting (combine weak learners -> always use the same model):
  + Adaboost
* Stacking :
  + Decision tree
  + Random forest
  + …

SKLEARN MODELS THAT PROVIDE A PREDICTION PROBABILITY

from sklearn.ensemble import **AdaBoostClassifier** # PROBABILITY

from sklearn.tree import **DecisionTreeClassifier** # PROBABILITY

from sklearn.neighbors import **RadiusNeighborsClassifier**

from sklearn.linear\_model import **RidgeClassifier**

from sklearn.ensemble import **GradientBoostingClassifier**

from sklearn.ensemble import **RandomForestClassifier** # PROBABILITY

from sklearn.discriminant\_analysis import **LinearDiscriminantAnalysis**

from sklearn.neighbors import **KNeighborsClassifier** # PROBABILITY

from sklearn.linear\_model import **LogisticRegression** # PROBABILITY

from sklearn.naive\_bayes import **GaussianNB** # PROBABILITY

from sklearn.ensemble import **ExtraTreesClassifier** # PROBABILITY

from sklearn.neighbors import **KNeighborsClassifier** # PROBABILITY

from sklearn.ensemble import **BaggingClassifier** # PROBABILITY

| Table 1-1 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| From the notebook “ missing value -> classifier selection “ -> report of the first 20 classifiers (name, accuracy, standard deviation) -> 10 K-fold cross-validation -> for models details see class dataTest.py. —> ENSEMBLING METHODS ALWAYS WIN! | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **TEST -> knn on balanced set (first balance and then impute) -> 6 min -> under-sampling** | | |  | **TEST -> knn on balanced set (first impute and then balance) -> undersampling** | | |  | **TEST - > mice inpute on all the data -> "col" 5 -> undersampling** | | |  | **TEST mice -> col 20 -> 6 minutes -> undersampling** | | |  | **TEST mice impute + SMOTE resampling to increase the training data set of pers -> over sampling** | | |  | **TEST mice impute + ADASYN resampling to increase the training data set of pers -> over sampling** | | |  | **TEST mice + TOMEKLINK -> undersampling -> NOT BALANCED** | | |
| **1** | bag10 | 58.92 | 3.93 |  | ada2 | 59.35 | 3 |  | bag7 | 60.97 | 2.81 |  | gb4 | 59.36 | 5.36 |  | ada4 | 83.22 | 2.57 |  | ada4 | 83.88 | 3.06 |  | ada7 | 66.5 | 3.47 |
| **2** | gb4 | 58.78 | 5.74 |  | gb2 | 59.08 | 4.31 |  | knn4 | 59.49 | 5.72 |  | bag8 | 59.21 | 4.31 |  | ada8 | 82.68 | 2.76 |  | et2 | 82.8 | 2.98 |  | bag11 | 66.31 | 2.1 |
| **3** | bag8 | 58.78 | 5.23 |  | bag8 | 59.08 | 6 |  | gb3 | 59.35 | 3.01 |  | ada4 | 59.1 | 5.74 |  | et3 | 82.28 | 2.6 |  | et3 | 82.6 | 3.05 |  | gb3 | 66.12 | 0.94 |
| **4** | gb2 | 58.49 | 4.08 |  | bag4 | 58.93 | 4.64 |  | ada4 | 59.08 | 4.22 |  | knn4 | 59.05 | 4.56 |  | et2 | 81.68 | 2.62 |  | ada8 | 82 | 2.58 |  | rf2 | 66.01 | 1.94 |
| **5** | knn5 | 58.48 | 6.95 |  | bag6 | 58.76 | 4.23 |  | gb4 | 59.07 | 5.72 |  | bag7 | 58.92 | 4.62 |  | et1 | 81.28 | 2.39 |  | et1 | 81.19 | 2.21 |  | ada4 | 65.92 | 1.79 |
| **6** | bag7 | 58.46 | 5.67 |  | knn5 | 58.63 | 6.21 |  | bag4 | 58.74 | 5.36 |  | bag9 | 58.77 | 5.26 |  | ada7 | 81.15 | 2.52 |  | ada7 | 80.79 | 3.01 |  | gb1 | 65.92 | 1.12 |
| **7** | et3 | 58.34 | 3.16 |  | bag11 | 58.34 | 5.89 |  | bag9 | 58.63 | 5.07 |  | knn5 | 58.64 | 6.92 |  | gb4 | 78.88 | 1.74 |  | ada3 | 78.78 | 3.45 |  | bag8 | 65.92 | 0.43 |
| **8** | bag9 | 58.34 | 5.38 |  | bag9 | 58.05 | 6.15 |  | gb2 | 58.61 | 6.73 |  | bag5 | 58.62 | 5.12 |  | bag7 | 78.81 | 2.55 |  | bag11 | 78.58 | 4.35 |  | bag9 | 65.92 | 0.87 |
| **9** | knn4 | 58.02 | 5.73 |  | bag10 | 58.05 | 6.46 |  | bag11 | 58.5 | 4.88 |  | ada6 | 58.49 | 4.96 |  | bag6 | 78.54 | 2.75 |  | gb2 | 78.44 | 3.01 |  | rf1 | 65.83 | 0.83 |
| **10** | bag11 | 57.76 | 4.8 |  | ada4 | 58.04 | 5.58 |  | bag6 | 58.47 | 3.5 |  | ada7 | 58.33 | 6.76 |  | bag11 | 78.41 | 2.03 |  | gb4 | 78.31 | 4.31 |  | bag10 | 65.82 | 1.46 |
| **11** | et2 | 57.59 | 5.31 |  | bag7 | 58.04 | 5.35 |  | bag3 | 58.31 | 4.45 |  | bag6 | 58.33 | 5.55 |  | ada3 | 78.27 | 2.81 |  | bag6 | 78.17 | 3.74 |  | et1 | 65.72 | 1.64 |
| **12** | ada2 | 57.16 | 5.56 |  | bag5 | 57.9 | 4.7 |  | bag5 | 58.31 | 4.82 |  | bag11 | 58.18 | 3.6 |  | bag5 | 78.01 | 2.38 |  | bag7 | 77.7 | 4.48 |  | et2 | 65.53 | 1.98 |
| **13** | ada6 | 57.16 | 5.82 |  | ada6 | 57.89 | 5.81 |  | knn1 | 58.18 | 6.86 |  | ada2 | 57.88 | 6.6 |  | bag10 | 77.07 | 2.51 |  | bag5 | 77.09 | 3.17 |  | gb4 | 65.43 | 1.69 |
| **14** | et1 | 57.01 | 5.36 |  | ada1 | 57.85 | 6.58 |  | knn5 | 58.04 | 5.91 |  | bag10 | 57.76 | 5.78 |  | bag4 | 76.67 | 2.54 |  | bag4 | 76.96 | 4.29 |  | ridge1 | 65.33 | 2.73 |
| **15** | knn1 | 56.97 | 6.94 |  | gb4 | 57.75 | 7.09 |  | bag10 | 57.75 | 3.36 |  | et3 | 57.61 | 5.3 |  | gb3 | 75.94 | 1.39 |  | rf3 | 76.63 | 2.9 |  | lr1 | 65.33 | 2.75 |
| **16** | ridge1 | 56.87 | 6.12 |  | ridge1 | 57.74 | 6.4 |  | gb1 | 57.74 | 4.79 |  | ada1 | 57.46 | 4.9 |  | gb2 | 75.86 | 2.33 |  | gb3 | 76.49 | 4.33 |  | ld1 | 65.14 | 2.56 |
| **17** | rf3 | 56.74 | 6.31 |  | ld1 | 57.74 | 6.4 |  | knn3 | 57.59 | 4.57 |  | rf3 | 57.46 | 5.23 |  | ada6 | 75.66 | 2.11 |  | bag10 | 75.49 | 3.24 |  | ada8 | 65.05 | 1.48 |
| **18** | ld1 | 56.72 | 5.9 |  | knn1 | 57.57 | 6.1 |  | bag8 | 57.47 | 5.65 |  | knn1 | 57.45 | 6.32 |  | rf3 | 75.4 | 2.26 |  | bag3 | 75.28 | 3.46 |  | dt2 | 64.95 | 3.14 |
| **19** | ada4 | 56.71 | 3.3 |  | ada3 | 57.44 | 5.27 |  | ada3 | 57.3 | 4.76 |  | gb3 | 57.16 | 5.47 |  | bag3 | 75.2 | 2.34 |  | ada2 | 75.22 | 3.51 |  | knn5 | 64.95 | 1.14 |
| **20** | ada3 | 56.59 | 5.19 |  | lr1 | 57.3 | 6.35 |  | et1 | 57 | 6.15 |  | knn3 | 57.02 | 4.45 |  | bag2 | 74.19 | 3.27 |  | ada6 | 73.33 | 4.04 |  | bag6 | 64.85 | 2.1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Attributes:

* feature\_importances\_. ## are sharable?

Methods :

* Predict -> return the label
* predict\_proba -> return the predicted probability
* predict\_log\_proba -> return class log-probabilities of the input samples X.
* decision\_path -> return the sequence of nodes followed in the tree

from sklearn.tree import **DecisionTreeClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **apply**(X[, check\_input]) | Returns the index of the leaf that each sample is predicted as. |
| **decision\_path**(X[, check\_input]) | Return the decision path in the tree |
| **fit**(X, y[, sample\_weight, check\_input, …]) | Build a decision tree classifier from the training set (X, y). |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X[, check\_input]) | Predict class or regression value for X. |
| **predict\_log\_proba**(X) | Predict class log-probabilities of the input samples X. |
| **predict\_proba**(X[, check\_input]) | Predict class probabilities of the input samples X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.ensemble import **AdaBoostClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **decision\_function**(X) | Compute the decision function of X. |
| **fit**(X, y[, sample\_weight]) | Build a boosted classifier from the training set (X, y). |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict classes for X. |
| **predict\_log\_proba**(X) | Predict class log-probabilities for X. |
| **predict\_proba**(X) | Predict class probabilities for X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |
| **staged\_decision\_function**(X) | Compute decision function of X for each boosting iteration. |
| **staged\_predict**(X) | Return staged predictions for X. |
| **staged\_predict\_proba**(X) | Predict class probabilities for X. |
| **staged\_score**(X, y[, sample\_weight]) | Return staged scores for X, y. |

from sklearn.linear\_model import **RidgeClassifier**

|  |  |
| --- | --- |
| **decision\_function**(X) | Predict confidence scores for samples. |
| **fit**(X, y[, sample\_weight]) | Fit Ridge regression model. |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class labels for samples in X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.ensemble import **GradientBoostingClassifier**

|  |  |
| --- | --- |
| **apply**(X) | Apply trees in the ensemble to X, return leaf indices. |
| **decision\_function**(X) | Compute the decision function of X. |
| **fit**(X, y[, sample\_weight, monitor]) | Fit the gradient boosting model. |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class for X. |
| **predict\_log\_proba**(X) | Predict class log-probabilities for X. |
| **predict\_proba**(X) | Predict class probabilities for X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |
| **staged\_decision\_function**(X) | Compute decision function of X for each iteration. |
| **staged\_predict**(X) | Predict class at each stage for X. |
| **staged\_predict\_proba**(X) | Predict class probabilities at each stage for X. |

from sklearn.ensemble import **RandomForestClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **apply**(X) | Apply trees in the forest to X, return leaf indices. |
| **decision\_path**(X) | Return the decision path in the forest |
| **fit**(X, y[, sample\_weight]) | Build a forest of trees from the training set (X, y). |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class for X. |
| **predict\_log\_proba**(X) | Predict class log-probabilities for X. |
| **predict\_proba**(X) | Predict class probabilities for X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.discriminant\_analysis import **LinearDiscriminantAnalysis**

|  |  |
| --- | --- |
| **decision\_function**(X) | Predict confidence scores for samples. |
| **fit**(X, y) | Fit LinearDiscriminantAnalysis model according to the given training data and parameters. |
| **fit\_transform**(X[, y]) | Fit to data, then transform it. |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class labels for samples in X. |
| **predict\_log\_proba**(X) | Estimate log probability. |
| **predict\_proba**(X) | Estimate probability. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |
| **transform**(X) | Project data to maximize class separation. |

from sklearn.neighbors import **KNeighborsClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **fit**(X, y) | Fit the model using X as training data and y as target values |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **kneighbors**([X, n\_neighbors, return\_distance]) | Finds the K-neighbors of a point. |
| **kneighbors\_graph**([X, n\_neighbors, mode]) | Computes the (weighted) graph of k-Neighbors for points in X |
| **predict**(X) | Predict the class labels for the provided data |
| **predict\_proba**(X) | Return probability estimates for the test data X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.linear\_model import **LogisticRegression** # PROBABILITY

|  |  |
| --- | --- |
| **decision\_function**(X) | Predict confidence scores for samples. |
| **densify**() | Convert coefficient matrix to dense array format. |
| **fit**(X, y[, sample\_weight]) | Fit the model according to the given training data. |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class labels for samples in X. |
| **predict\_log\_proba**(X) | Log of probability estimates. |
| **predict\_proba**(X) | Probability estimates. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |
| **sparsify**() | Convert coefficient matrix to sparse format. |

from sklearn.naive\_bayes import **GaussianNB** # PROBABILITY

|  |  |
| --- | --- |
| **fit**(X, y[, sample\_weight]) | Fit Gaussian Naive Bayes according to X, y |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **partial\_fit**(X, y[, classes, sample\_weight]) | Incremental fit on a batch of samples. |
| **predict**(X) | Perform classification on an array of test vectors X. |
| **predict\_log\_proba**(X) | Return log-probability estimates for the test vector X. |
| **predict\_proba**(X) | Return probability estimates for the test vector X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.ensemble import **ExtraTreesClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **apply**(X) | Apply trees in the forest to X, return leaf indices. |
| **decision\_path**(X) | Return the decision path in the forest |
| **fit**(X, y[, sample\_weight]) | Build a forest of trees from the training set (X, y). |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class for X. |
| **predict\_log\_proba**(X) | Predict class log-probabilities for X. |
| **predict\_proba**(X) | Predict class probabilities for X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.neighbors import **KNeighborsClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **fit**(X, y) | Fit the model using X as training data and y as target values |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **kneighbors**([X, n\_neighbors, return\_distance]) | Finds the K-neighbors of a point. |
| **kneighbors\_graph**([X, n\_neighbors, mode]) | Computes the (weighted) graph of k-Neighbors for points in X |
| **predict**(X) | Predict the class labels for the provided data |
| **predict\_proba**(X) | Return probability estimates for the test data X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |

from sklearn.ensemble import **BaggingClassifier** # PROBABILITY

|  |  |
| --- | --- |
| **decision\_function**(X) | Average of the decision functions of the base classifiers. |
| **fit**(X, y[, sample\_weight]) | Build a Bagging ensemble of estimators from the training set (X, y). |
| **get\_params**([deep]) | Get parameters for this estimator. |
| **predict**(X) | Predict class for X. |
| **predict\_log\_proba**(X) | Predict class log-probabilities for X. |
| **predict\_proba**(X) | Predict class probabilities for X. |
| **score**(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| **set\_params**(\*\*params) | Set the parameters of this estimator. |