CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Machine Learning

Machine Learning

Lecture: Feature Selection

Ted Scully

Data Pre-processing for Scikit Learn

- Dealing with Outliers (Optional)
- Dealing with Missing Values
- Handling Categorical Data
- Scaling Data
- Handling Imbalance
- **Feature Selection**
- Dimensionality Reduction

Feature Selection

- Feature Selection is the process of ranking or quantifying the contribution of each feature in a dataset in helping to predict the target regression value of class.

- There are a broad range of feature selection techniques.

- We will look at three methods of feature selection supported by Sci-kit Learn:

  - Univariate Feature Selection
  - Tree-Based Feature Selection
  - Greedy Backward Feature Selection

Univariate Feature Selection

- Scikit learn has a number of uni-variate approaches to feature selection, which examine the relationship between the **features** and the **target class**.

- The main univariate feature selection tools in sklearn are: SelectPercentile and SelectKBest. Both of these functions allow you to run statistical tests between the features and the target class.

- **sklearn.feature_selection.SelectKBest(score_func=<function f_classif>, k=10)**

- **Parameters:**
  - score_func : sklearn provides a number of score functions.
    - f_regression (used only for numeric targets and based on linear regression performance)
    - F_classif (used only for categorical targets and based on ANOVA statistical test)
  - k : int or "all", optional, default=10 Number of top features to select.

Univariate Feature Selection

```
from sklearn import datasets
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_regression

boston = datasets.load_boston()
X = boston.data
y = boston.target

selector = SelectPercentile(f_regression, percentile=25)
selector.fit(X,y)

for n,s in zip(boston.feature_names, selector.scores_):
   print ("Score : ", s, " for feature ", n)
```

```
Score :  88.1512417809  for feature  CRIM
Score :  75.257642299  for feature  ZN
Score :  153.954883136  for feature  INDUS
Score :  15.9715124204  for feature  CHAS
Score :  112.59148028  for feature  NOX
Score :  471.846739876  for feature  RM
Score :  83.4774592192  for feature  AGE
Score :  33.5795703259  for feature  DIS
Score :  85.9142776698  for feature  RAD
Score :  141.761356577  for feature  TAX
Score :  175.105542876  for feature  PTRATIO
Score :  63.0542291125  for feature  B
Score :  601.61787111  for feature  LSTAT
```

- These univariate tests have certain limitations. Some features may be **unfairly penalized**. Univariate feature selection can negatively bias against categorical data types or binary data types (especially categorical features with a limited range).
- Also univariate selection techniques typically only measure **linear** relationships and clearly they cannot capture **interaction** between features.

Tree Based Feature Selection

- Decision trees work by reducing the level of **uncertainty** or **impurity** in the data and can use one of a range of metrics to do this (gini, entropy, etc).

- They select the feature that provides the largest reduction in overall uncertainty.

- Also for each feature we can quantify (measure) the reduction in uncertainty.

- **Tree based feature selection** builds **many trees** and calculates the **average reduction in uncertainty** achieved by each feature across all the trees and uses this as a means of feature ranking.

- The features with the largest reduction have the highest impact (the most important features).

Decision Tree

- When building a decision tree, the algorithm will need to decide on what feature to use for each node in the tree. It examines all features and selects the feature that provides the largest reduction in overall uncertainty.
- Put another way it select the feature that gives the greatest insight into the class.

Rain	Swell	Surf
Yes	Big	Yes
Yes	Small	No
No	Small	No
No	Big	Yes

Ensemble Learning

- In machine learning, ensemble methods utilize/**combine the results** from a number of learning algorithms to obtain a better predictive performance than could be obtained from any single one of the learning algorithms learning algorithms

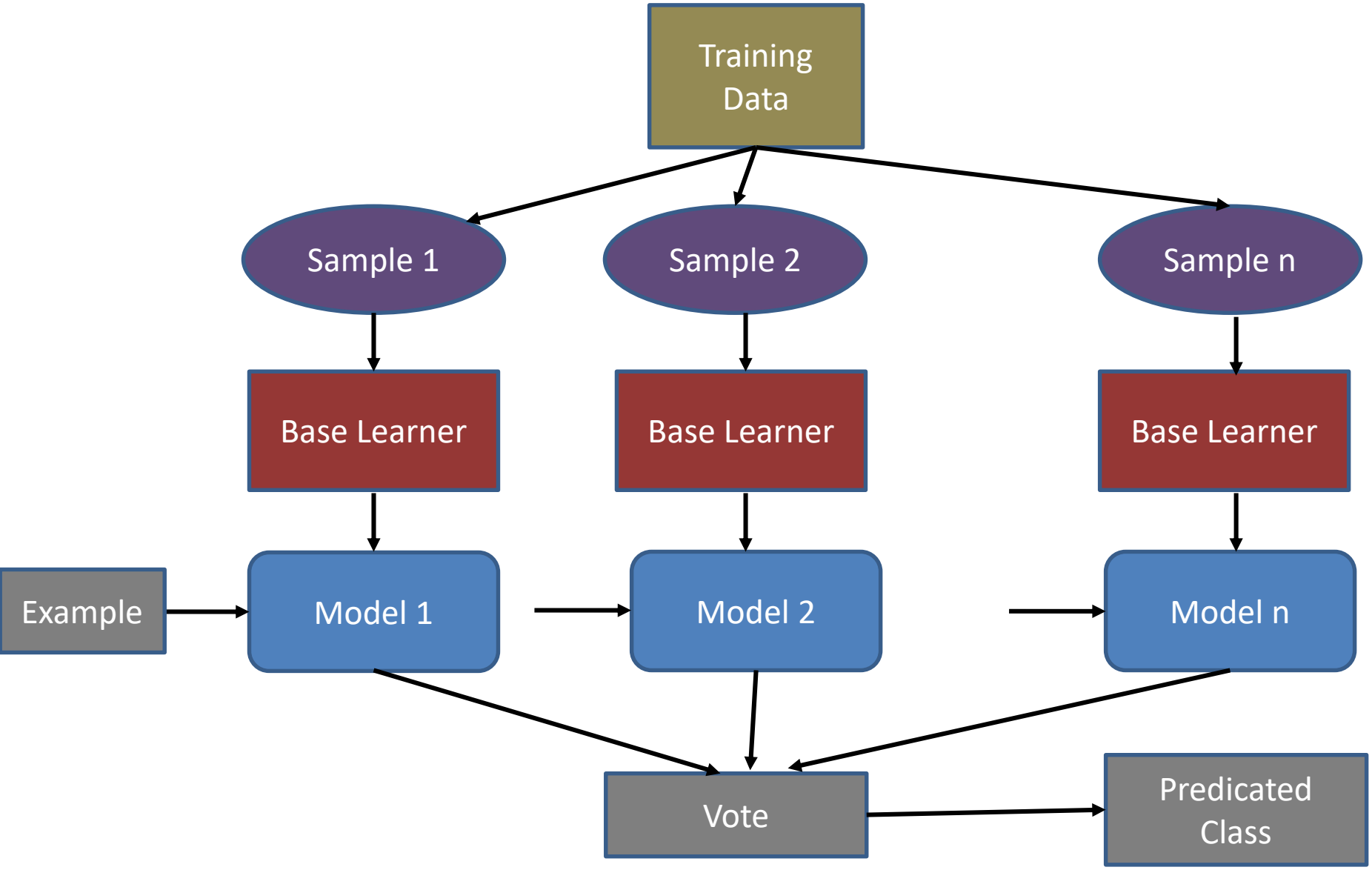
- **Instead of learning one model, we learn several** and combine them (vote for classification, average for regression). Such combinations are known as model ensembles.
  - Note the base learners could be the same model or different models

- They are among the most powerful techniques in machine learning. Typically improve accuracy.

Introduction to Bagging

- **Bagging, short for "bootstrap aggregating"** is a simple but effective technique that creates different models on **random samples** of the original data set.
  - These samples are taken uniformly with replacement and are known as **bootstrap samples**.
- Learn one model for each sample
- Because samples are taken at random with replacement each sample will normally contain **duplicates** and hence some of the original data will be missing from your sample.
- This technique has shown that **statistically each of the samples look very similar** (variance, mean,etc)

Introduction to Bagging



Decision Trees and Subspace Sampling

- Bagging is particularly effective with **decision tree models**.
- Decision trees can be **very sensitive to the underlying data**. Changing a small percentage of the training data could produce a very different looking tree (in particular the lower levels).
- Other methods can also be used to inject additional diversity into the decision trees. One common method is when selecting a node for a tree using a **random subset of the features**. For each node we build in the tree we base on decision on a different subset of features each time.
- This is sometimes called **random subspace sampling**.

Random Forests

- Random Forest is an ensemble techniques that combines both bagging and subspace sampling.

Input: data set D; ensemble size T; subspace dimension d

Function RandomForest (D, T, d)

for t=1 to T **do**:

- Build a bootstrap sample D_t from D by sampling |D| data points with replacement
- buildRandomTree(D_t , d)

Function BuildRandomTree(D_t , d)

- **At each node:**
  - f = Select d features at random and reduce dimensionality of D_t accordingly
  - Split on best feature in f

return learned tree

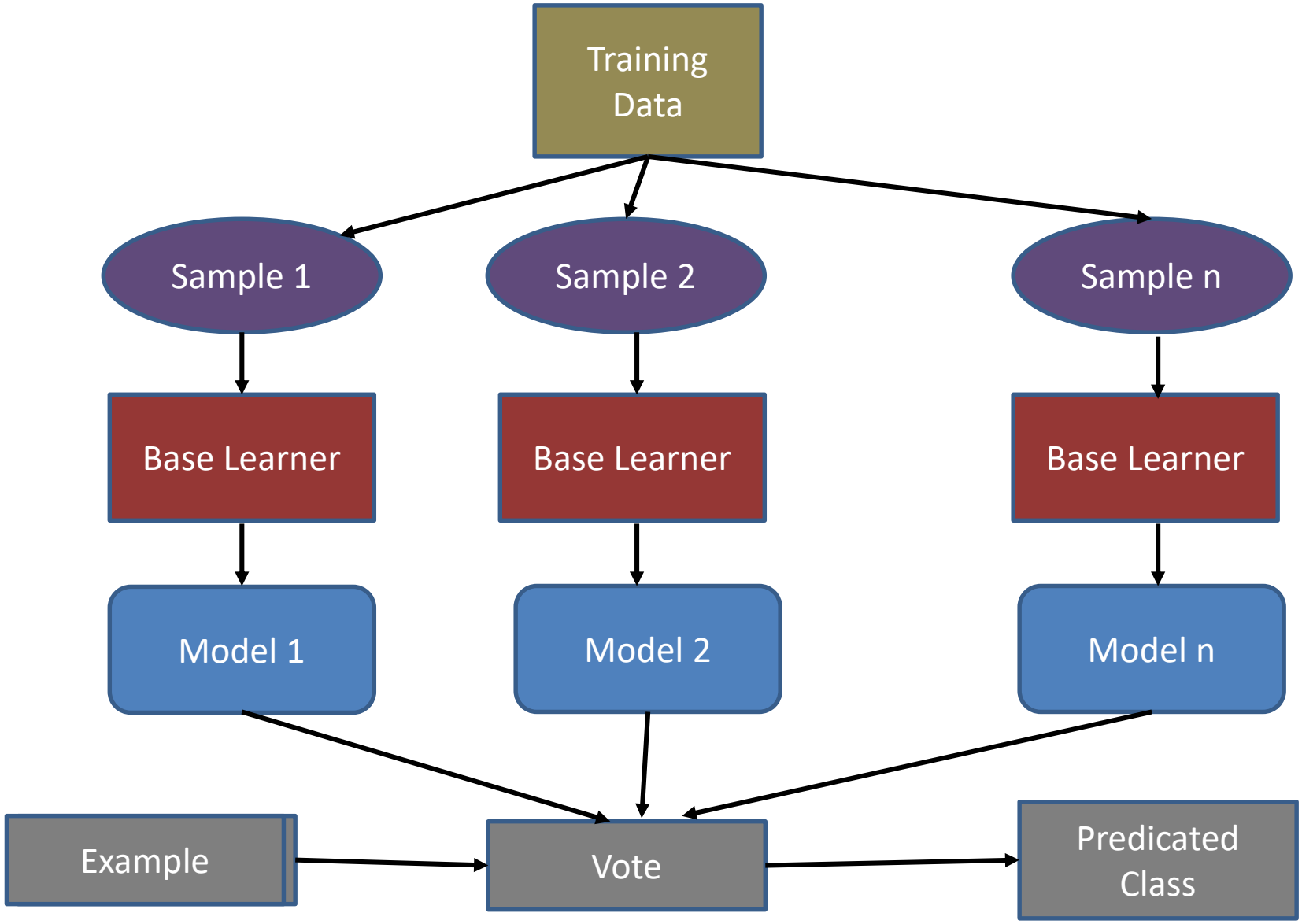
Back to Feature Selection

- When **building a single decision tree** every time it creates a new split it looks at the data available for that split and determines which feature provides the **greatest reduction in uncertainly**.

- The RF will build many trees (you control the number of trees built). It can then return the **feature importance** of each feature by **averaging the reduction in uncertainty** achieved by each feature across all trees generated.

- The default measure of uncertainty used in a RF is the Gini index.

Introduction to Bagging



Attributes:

estimators_ : list of DecisionTreeClassifier

The collection of fitted sub-estimators.

classes_ : array of shape = [n_classes] or a list of such arrays

The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).

n_classes_ : int or list

The number of classes (single output problem), or a list containing the number of classes for each output (multi-output problem).

n_features_ : int

The number of features when `fit` is performed.

n_outputs_ : int

The number of outputs when `fit` is performed.

feature_importances_ : array of shape = [n_features]

The feature importances (the higher, the more important the feature).

Tree – Based Feature Selection

- Tree-based ensemble methods such as **random forests** can be effectively used for feature selection for classification problems.

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000,
                           n_features=10,
                           n_informative=4,
                           n_redundant=0,
                           n_repeated=0,
                           n_classes=2,
                           random_state=0,
                           shuffle=False)

# Build a forest and compute the feature importance
forest = RandomForestClassifier(n_estimators=250, random_state=0)
forest.fit(X, y)

importances = forest.feature_importances_

for index in range(len(X[0])):
    print ("Importance of feature ", index, "is", importances[index])
```

Feature ranking:
1. feature 0 (0.206926)
2. feature 1 (0.176586)
3. feature 3 (0.170126)
4. feature 2 (0.151502)
5. feature 7 (0.050760)
6. feature 5 (0.050182)
7. feature 9 (0.049881)
8. feature 4 (0.049276)
9. feature 8 (0.047586)
10. feature 6 (0.047175)

Greedy Feature Selection

- Greedy feature selection automatically reduces the number of features involved in a learning model based on their effective contribution to the overall accuracy performance of the algorithm.

- The **RFECV** class provides a greedy backward search that can provide you with information on the number of useful features, points them out to you and automatically transform the X data, into a reduced variable set, as shown on the next slide.

- Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model) or that builds a feature importance array , recursive feature elimination (RFE) selects features by recursively considering smaller and smaller sets of features.
  - First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a **coef_** attribute or through a **feature_importances_** attribute.
  - Then, the least important feature is pruned from current set of features.
  - That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached (or until all features have been ranked).

Greedy Feature Selection

- The basic syntax is as follows:
- ***RFECV(estimator, step=1, cv=None, scoring=None)***

- Note you should select an estimator that should provide information about feature importance through a **coef_ attribute or feature_importance**

- Returns
  - **n_features_** : number of selected features (these are the number of features that gave the strongest result)
  - **ranking_** : feature ranking such that **ranking_[i]** corresponds to the ranking position of the **i-th** feature (Best features ranked 1)
  - **support_** : Boolean array containing a true if feature is selected (as rank 1), false otherwise
  - **grid_scores_** : array of shape [n_subsets_of_features]
  - The scores such that grid_scores_[i] corresponds to the score of the i-th subset of features.

```
from sklearn import model_selection
from sklearn.feature_selection import RFECV
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn import linear_model
import matplotlib.pyplot as plt

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000, n_features=25, n_informative=3,
                           n_redundant=2, n_repeated=0, n_classes=8,
                           n_clusters_per_class=1, random_state=0)

decTree = DecisionTreeClassifier()
scores = model_selection.cross_val_score(decTree, X, y, cv=10)
print ('Initial Result',scores.mean())

estimator = linear_model.LogisticRegression(multi_class='auto', solver ='lbfgs')
rfecv = RFECV(estimator, cv=10)
rfecv.fit(X, y)

# optimal number of features
print (rfecv.n_features_)

# ranking of each feature
print (rfecv.ranking_)
```

Create the classification dataset and assess the accuracy of a DecisionTree Model.

Initial Result 0.723

```
from sklearn import model_selection
from sklearn.feature_selection import RFECV
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn import linear_model
import matplotlib.pyplot as plt

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000, n_features=25, n_informative=3,
                           n_redundant=2, n_repeated=0, n_classes=8,
                           n_clusters_per_class=1, random_state=0)

decTree = DecisionTreeClassifier()
scores = model_selection.cross_val_score(decTree, X, y, cv=10)
print ('Initial Result',scores.mean())

estimator = linear_model.LogisticRegression(multi_class='auto', solver ='lbfgs')
rfecv = RFECV(estimator, cv=10)
rfecv.fit(X, y)

# optimal number of features
print (rfecv.n_features_)

# ranking of each feature
print (rfecv.ranking_)
```

Execute RFECV to rank features. Print the optimal number of features and the final numerical ranks of the features.

```
3
[ 6  1 22 17  8  9 10  1  2 16 19 11 12 14 15  4  5 23  1 13
20 18  3  7 21]
```

```
# select highest ranked features and build a new model
X = X[ : , rfecv.support_ ]
decTree = DecisionTreeClassifier()
scores = model_selection.cross_val_score(decTree, X, y, cv=10)
print ('Result after feature selection: ',scores.mean())
```

Use the Boolean mask array support_ to access the features that are ranked highest (as rank 1).
Rebuild the decision tree and assess the accuracy on the reduced dataset.

Result after feature selection: 0.769