# 5 Machine Learning

**Chapter Description** This is chapter is intended to provide participants with good understanding and practical experience in SVM and Ensemble techniques for classification.

**Chapter Objective**

By the end of this chapter, the participants will be able to:

* Implementation of SVM and Ensemble techniques to solve practical classification problem.

**Enabling Objectives**

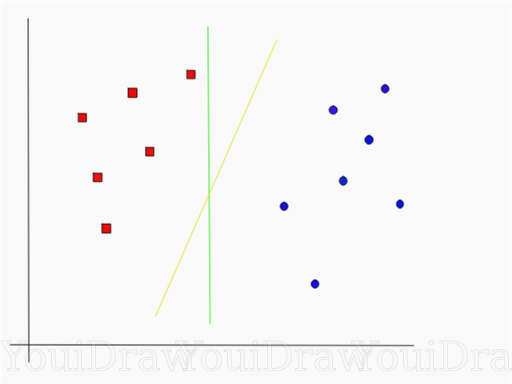
After completing this session, the participants will be able to:

* Describe SVM classification techniques.
* Differentiate between Linear and Non-linear SVM techniques
* Implement SVM technique using Python to solve real problems.
* Describe Ensemble techniques.
* Describe different Ensemble Techniques Bagging, Stacking and Boosting
* Implement Bagging, Stacking and Boosting techniques to solve real health problems.

## 5.1 Support Vector Machine(SVM)

Support Vector Machine (SVM) is a well-known and powerful Supervised learning technique used for classification, regression and even outlier and anomaly detection.

SVM takes input features of the data (training data) and generates a multi-dimensional hyperplane that best classify the boundaries, as can be seen from figure XX.

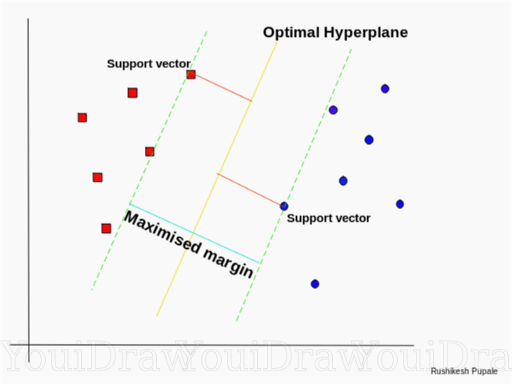


Figure

SVM constructs a [hyperplane](https://en.wikipedia.org/wiki/Hyperplane)  that are boundaries of data points in a [high-](https://en.wikipedia.org/wiki/High-dimensional_space) or infinite-dimensional space. Therefore, it works by maximizing the distance between support vectors, i.e. *margin*, since in general the larger the margin, the lower the [generalization error](https://en.wikipedia.org/wiki/Generalization_error) (less predicting power) of the classifier. Therefore, we require the largest possible margin to avoid high variance.

Support vector machines (SVMs) were originally designed for binary classification, are large margin classifiers that try to separate instances of different classes with the maximum margin hyperplane.

SVM is particularly suited complex small to medium datasets.



*Support Vectors and Hyperplane*

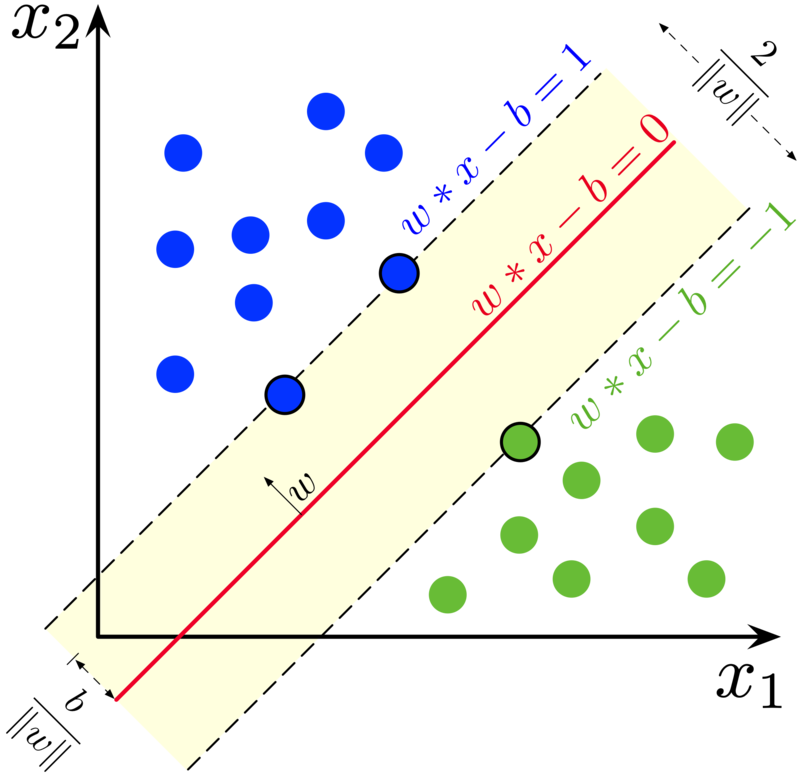
Hyperplane is a decision boundary, or a decision boundary plane for the case of three dimensional space, that distinguishes between data points. If the number of input features is two, the hyperplane is a Decision boundary line. For the case of three input features, hyperplane shall be boundary plane. It is difficult to imagine for more than three input features.

Support Vectors are data points close to the hyperplane that influence the orientation and position of the hyperplane. Therefore, the support vectors are used to maximize the distance between classification regions.

Fig

**5.1. Linear Support Vector Machines**

In some situations, the data points to be classified are clear and distinct, and can be classified using a straight line. While using linear SVM, it is assumed that the data are linearly separable. In such cases, simple linear function of Hyperplane and decision boundaries would be sufficient for classification, as can be seen in fig .



Fig

**Highlights in how SVM works (Optional)**

As mention earlier, SVM works by creating hyperplane and decision boundaries, and maximizing margin (w).

The equation of line can be given as

Where, w is the weight and

b is the bias

As we are considering a multi-dimensional vector space for a data with multiple input features, weights, biases, x and y can be transposed and stacked into an array. Hence,

)

Perpendicular distance between a point and a line can be given as

Similarly, distance between hyperplane and a point is given as

The updated weight (w’ ) that maximizes the margin can be obtained by

()

The above equation () returns weight that correspond to the maximum margin size for the minimum (Closest point). Argmax function is short for argument maximum, which is a function that returns the argument corresponding to the maximum outcome.

Now, if the point is projected to the positive side of the plane,

By the same token, if the point is projected to the negative side of the hyperplane,

Since Classification problem is considered, the product of actual and predicted value yields positive results for correct predictions, whereas negative for wrong predictions as shown in the equation below.

The distance of the closest point can be rescaled to 1 for convenience as

= 1

Again, for convenience, the argmax function can be converted in to minimization problem so that we obtain convex function.

Equation () can then be modified as

The first part of equation is the main equation that gives the maximum.

*Case 1:* **Perfect Classification of Primal Form**

In the case perfect classification, the following condition holds true for all values of n.

(())

*Case 2:* **Non Perfect Classification of Primal Form (Slight Misclassification)**

A model that perfectly classifies is not desired as it can cause overfitting and may not generalize well. Therefore, it recommended to introduce some Bias to reduce the Variance (Bias-Variance tradeoff).

(())

Introducing a Slack variable and hyper-parameter value C , we obtain,

(())

Practically, Sci-kit learn is used to solve linear SVM. The hyper-parameter we saw earlier, C, is used in writing an SVM code using the SVC classifier. A larger C value can result in overfitting and our model will fail to generalize. On the other hand, small value of C result in higher bias. It is always important to maintain balance between Bias and Variance by tuning the C hyper-parameter.

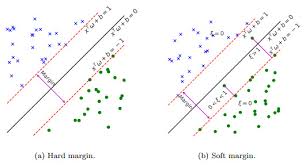
**Soft Margin Classification**

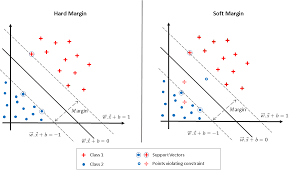
If we strictly implement that all instances shall be at and within the decision boundary, this type of classification is called Hard Margin Classification. Outliers are not allowed in Hard Margin Classification*.*

Hard margin is not recommended for quite some reasons. One is that it only works well for linearly separable datasets. It is also sensitive to outliers. Also in some situations, it is impossible to find a margin if support vectors are very close to each other.

To avoid this, Soft Margin classification, a more flexible classification technique is used. It keeps the balance between keeping the margin wide and limiting outliers not only within the margin, but also in the wrong side.

*But in the case of Soft Margin Classification, in the process of training some outlier are allowed to get within the decision boundaries*





**Error Metrics of Linear SVM**

In most cases, Hinge loss metrics is used for binary classification problem and is given as

L(t) = max{0, 1 − t} where t *=*

We can see here that, t shall equal 0 if the classification is wrong. This is because t1 for attribute on the right side of the Hyperplane and t<1 if it is on classified wrong.

**Practical of Linear SVM**

As was mentioned earlier, SVM is a powerful tool for classification problem, especially for Binary Classification. By SVM, we implicitly are referring to soft margin SVM.

SVM basically follows a Machine Learning Life Cycle. Once preprocessed, and SVM model is selected, the training and testing procedure continues.

Train test split shall be performed randomly, let us say 70% training set, 15% validation (Dev\_set) and 15% test set for the whole data, as we mostly use SVM for medium sized dataset. We first train the model using the 70% training set. We then validate whether the model is working using the 15% Validation set. We compare the training and validation set performance. If the training error is very small compared to validation set, then we can say the model has over fit and not able to generalize. If the training set accuracy is very low, the model is considered to have a high bias. Therefore, we must fine tune the hyper-parameter, C in case of SVM. Then once we obtain a very good train-validation accuracy combination is obtained, we once again combine Training and Validation set and train it and obtain final accuracy.

We then test the model and test it using the test set and see the power of generalization.

Another alternative for this procedure is to use Cross-Validation techniques. We shall be using k-fold cross validation technique for activities within this section.

The common error metric used in SVM especially for Binary Classification problem is the Hinge metric described previously.

*Activity*

**Non Linear SVM**

Previously, we have seen Linear SVM can be applies to linearly separable data. But there come many times where we encounter Non-linear data. In some cases, adding more polynomial feature can transform nonlinear data to linearly-separable.

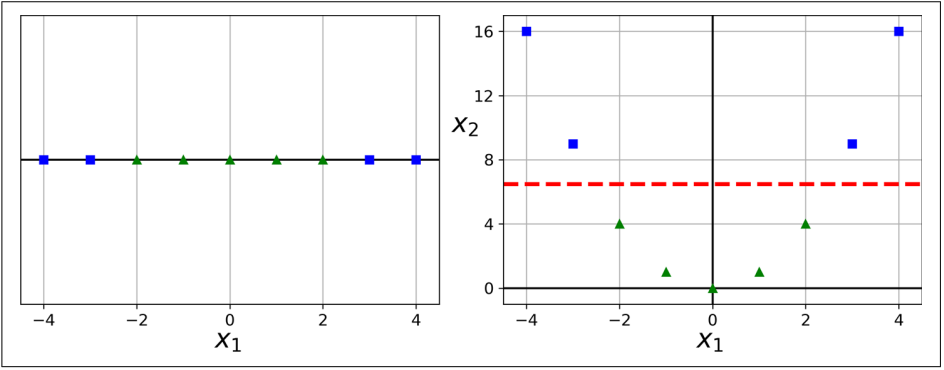


Fig.

As can be seen from the figure, the values on the left are linearly inseparable. Simple transformation is applied by squaring the values X2. By plotting X1 and X2, it can simply be transformed to linearly separable instances.

**Outline**

* Basics of SVM
* Linear SVM
* Nonlinear SVM (Polynomial and Radial Basis Function)

## 5.2 Ensemble Machine Learning

**Outline**

* Bagging
* Staking
* Boosting
  + XGBoost

Ensemble methods train multiple learners to solve the same problem. In contrast to ordinary learning approaches which try to construct one learner from training data, ensemble methods try to construct a set of learners and combine them. Ensemble learning is also called committee-based learning, or learning multiple classifier systems. An ensemble contains a number of learners called base learners. Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e., learners of the same type, leading to homogeneous ensembles, but there are also some methods which use multiple learning algorithms to produce heterogeneous learners, i.e., learners of different types, leading to heterogeneous ensembles.

The generalization ability of an ensemble is often much stronger than that of base learners. Actually, ensemble methods are appealing mainly be- cause they are able to boost weak learners which are even just slightly better than random guess to strong learners which can make very accurate predictions. So, base learners are also referred to as weak learners. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability or robustness over a single estimator.

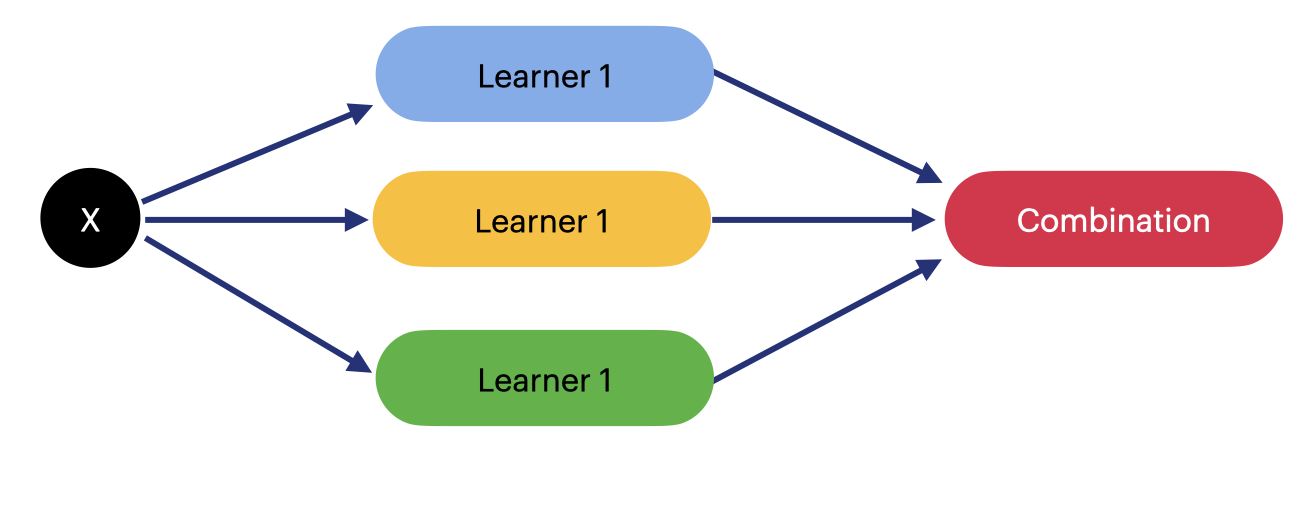


FIGURE 5.1: Ensemble architecture

### 5.2.1 Ensemble Methods

The three most popular methods for combining the predictions from different models are:

* Bagging. Building multiple the same type models from different subsamples of the training dataset.
* Boosting. Building multiple the same type models each of which learns to fix the prediction errors of a prior model in the sequence of models.
* Stacking(Voting). Building multiple different models and simple statistics (like calculating the mean) are used to combine predictions.

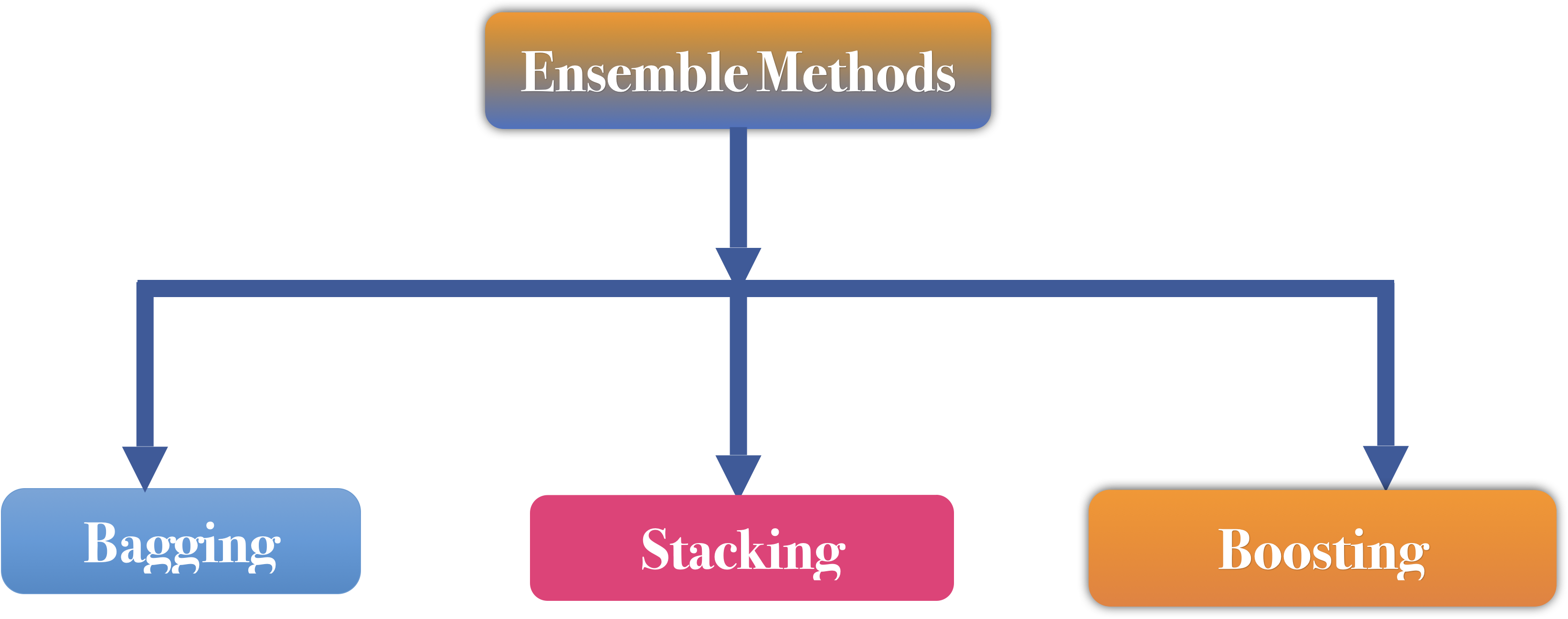


FIGURE 5.2: The three most popular methods for combining the predictions

### 5.2.2 Bagging Ensemble

Bootstrap Aggregation (or Bagging) involves taking multiple samples from your training dataset (with replacement) and training a model for each sample. The final output prediction is averaged across the predictions of all of the sub-models. Some of different bagging models to be be used for aggregating includes:

* Bagged Decision Trees.
* Random Forest.
* Extra Trees.

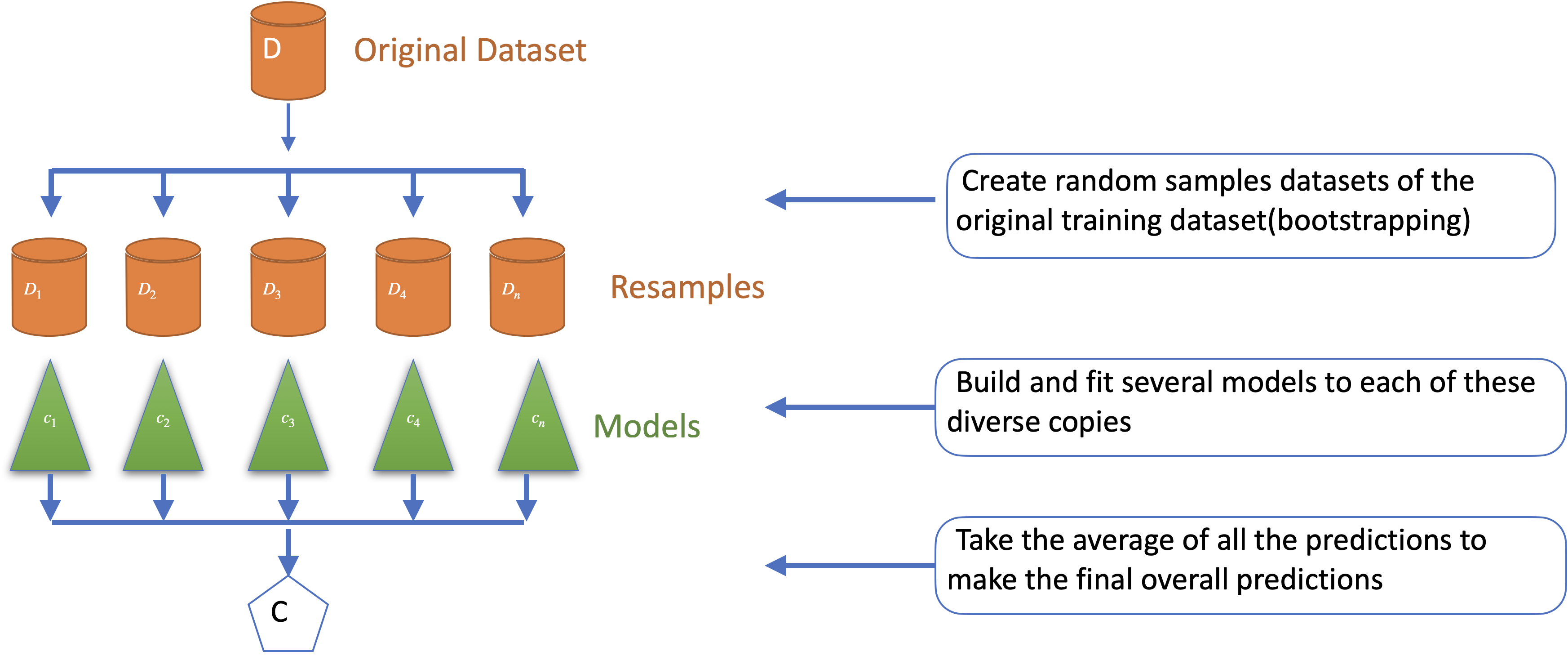


FIGURE 5.3: Bagging ensemble method

The bootstrap method goes as follows. Let there be a sample $X$ of size $N$. We can make a new sample from the original sample by drawing $N$ elements from the latter randomly and uniformly, with replacement. In other words, we select a random element from the original sample of size $N$ and do this $N$ times. All elements are equally likely to be selected, thus each element is drawn with the equal probability $\frac{1}{N}$.

Let’s say we are drawing balls from a bag one at a time. At each step, the selected ball is put back into the bag so that the next selection is made equiprobably i.e. from the same number of balls . Note that, because we put the balls back, there may be duplicates in the new sample. Let’s call this new sample .

By repeating this procedure times, we create *bootstrap samples* . In the end, we have a sufficient number of samples and can compute various statistics of the original distribution.

#### 5.2.2.1 Bagged Decision Trees

Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning. In the example below is an example of using the BaggingClassifier with the Classification and Regression Trees algorithm.

# Bagged Decision Trees for Classification  
import pandas as pd  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.ensemble import BaggingClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.datasets import load\_breast\_cancer  
df = load\_breast\_cancer()  
X = df.data  
y = df.target  
seed = 7  
kfold = KFold(n\_splits=10, random\_state=seed,shuffle=True )  
cart = DecisionTreeClassifier()  
num\_trees = 100  
model = BaggingClassifier(base\_estimator=cart,  
n\_estimators=num\_trees, random\_state=seed)   
results = cross\_val\_score(model, X, y, cv=kfold)  
print(results.mean())

The Bagged Decision Trees Classifier provides a mean estimate of classification accuracy

0.963095238095238

#### 5.2.2.2 Random Forest

Random Forests is an extension of bagged decision trees. Samples of the training data-set are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of each tree, only a random subset of features are considered for each split. You can construct a Random Forest model for classification using the RandomForestClassifier class

# Random Forest Classification  
import pandas as pd  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.ensemble import RandomForestClassifier # RF classifier  
from sklearn.datasets import load\_breast\_cancer  
df = load\_breast\_cancer()  
X = df.data  
y = df.target  
seed = 7  
max\_features = 3  
kfold = KFold(n\_splits=10, random\_state=7,shuffle=True)  
model = RandomForestClassifier(n\_estimators=num\_trees,  
max\_features=max\_features)   
results = cross\_val\_score(model, X, y, cv=kfold)  
print(results.mean())

The Random Forest Classifier provides a mean estimate of classification accuracy

0.9613408521303258

#### 5.2.2.3 Extra Trees

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset. You can construct an Extra Trees model for classification using ExtraTreesClassifier class.

# Extra Trees Classification  
import pandas as pd  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.ensemble import ExtraTreesClassifier # ETC classifier  
from sklearn.datasets import load\_breast\_cancer  
df = load\_breast\_cancer()  
X = df.data  
y = df.target  
seed = 7  
max\_features = 3  
num\_trees = 100  
max\_features = 7  
kfold = KFold(n\_splits=10, random\_state=7,shuffle=True)  
model = ExtraTreesClassifier(n\_estimators=num\_trees, max\_features=max\_features)   
results = cross\_val\_score(model, X, y, cv=kfold)  
print(results.mean())

The Random Forest Classifier provides a mean estimate of classification accuracy

0.9683897243107769

**Activity 5.2.2 Individual Reflection on bagging ensemble**

**Question**

Comparing the classification result of section 5.2.2.1,5.2.2.2 and 5.2.2.3, what classifier result a better classification accuracy?

**Time allowed:** 5Minutes

#### 5.2.2.4 Model Bias

### 5.2.3 Stacking(Voting) Ensemble

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms. It works by first creating two or more standalone models from your training data-set. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data. The predictions of the sub-models can be weighted, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from sub-models, but this is called stacking (stacked aggregation) and is currently not provided in scikit-learn. You can create a voting ensemble model for classification using the VotingClassifier class. The code below provides an example of combining the predictions of logistic regression, classification and regression trees and support vector machines together for a classification problem.

# Vote(Stacking) Classification  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
from sklearn.ensemble import VotingClassifier  
from sklearn.datasets import load\_breast\_cancer  
sns.set()  
## load breast cancer datset  
df = load\_breast\_cancer()  
X = df.data  
y = df.target  
  
# prepare models  
models = []  
models.append(('LR', LogisticRegression()))  
models.append(('CART', DecisionTreeClassifier()))  
models.append(('SVM', SVC()))  
  
# create the ensemble model  
ensemble = VotingClassifier(models)  
  
# evaluate each model in turn  
results = []  
names = []  
scoring = 'accuracy'  
for name, model in models:  
 kfold = KFold(n\_splits=10, random\_state=7,shuffle=True)  
 cv\_results = cross\_val\_score(ensemble, X, y, cv=kfold, scoring=scoring)   
 results.append(cv\_results)  
 names.append(name)  
 msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)  
# boxplot algorithm comparison  
fig = plt.figure()   
fig.suptitle('Algorithm Comparison')   
ax = fig.add\_subplot(111)   
plt.boxplot(results)   
ax.set\_xticklabels(names)   
plt.savefig('fig/stacking\_result.png')  
plt.show()

The Stacking Classifiers of Logistic Regression, Decision Tree and Support Vector provide a mean estimate of classification accuracy 0.952569

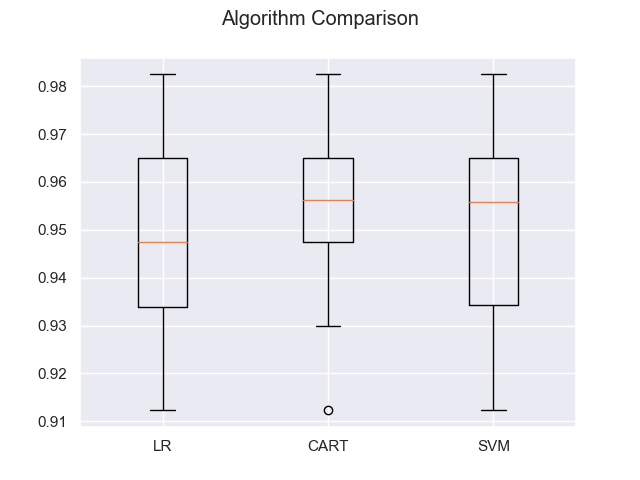


FIGURE 5.4: The stacking ensemble result box-plot

**Activity 5.2.2 Individual Reflection on bagging and stacking ensemble**

**Question**

Comparing the classification result of bagging section 5.2.2 and voting section 5.2.3, which ensemble is performing better for our problem?

**Time allowed:** 5Minutes

### 5.2.4 Boosting Ensemble

The term boosting refers to a family of algorithms that are able to convert weak learners to strong learners. Intuitively, a weak learner is just slightly better than random guess, while a strong learner is very close to perfect performance.

#### 5.2.4.1 Gradient boosting

Gradient boosting is one of the most powerful techniques for building predictive models that combines gradient descent and boosting. Instead of computing the overall true gradient explicitly, gradient boosting aims to approximate the true gradient with a weak learner. A weak learner is defined as one whose performance is at least slightly better than random chance. These ideas built upon Leslie Valiant’s work on distribution free or Probability Approximately Correct (PAC) learning, a framework for investigating the complexity of machine learning problems. Hypothesis boosting was the idea of filtering observations, leaving those observations that the weak learner can handle and focusing on developing new weak learns to handle the remaining difficult observations.

After completing this section, you will able to describe: \* The origin of boosting from learning theory and AdaBoost. \* How gradient boosting works including the loss function, weak learners and the additive model. \* How to improve performance over the base algorithm with various regularization schemes.

### 5.2.5 Adaptive Boosting

The first realization of boosting that saw great success in application was Adaptive Boosting(AdaBoost). The weak learners in AdaBoost are decision trees with a single split, called decision stumps for their shortness. AdaBoost works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns. ***This means that samples that are difficult to classify receive increasing larger weights until the algorithm identifies a model that correctly classifies these samples***.

AdaBoost and related algorithms were recast in a statistical framework first by Breiman calling them ARCing algorithms.Arcing is an acronym for Adaptive Reweighting and Combining. Each step in an arcing algorithm consists of a weighted minimization followed by a recomputation of **the classifiers** and **weighted input**. Gradient boosting( gradient tree boosting) is a numerical optimization problem where the objective is to minimize the loss of the model by adding weak learners using a gradient descent like procedure. It involves three elements: 1. A loss function to be optimized. 2. A weak learner to make predictions. 3. An additive model to add weak learners to minimize the loss function.

#### 5.2.5.1 Extreme Gradient Boosting(XGBoost)

XGBoost is the leading model for working with standard tabular data (the type of data you store in Pandas DataFrames, as opposed to data like images and videos. It has recently been dominating applied machine learning competitions for structured or tabular data. It is an implementation of gradient boosted decision trees designed for speed and performance. The implementation of the model supports the features of the scikit-learn implementations, with new additions like regularization. Three main forms of gradient boosting supported are:

* Gradient Boosting algorithm also called gradient boosting machine including the learning rate.
* Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.
* Regularized Gradient Boosting with both and regularization.

To reach peak accuracy, XGBoost models require more knowledge and model tuning than techniques like Random Forest. XGBoost is an implementation of the Gradient Boosted Decision Trees algorithm (scikit-learn has another version of this algorithm, but XGBoost has some technical advantages.) What is Gradient Boosted Decision Trees? We’ll walk through a diagram.

**Activity 5.2.5.1 Individual Guided Practice on Regularization**

**Question**: Discuss in a group of five about and .

**Time Allowed** : 15 Minutes

# Vote(Stacking) Classification  
import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
import seaborn as sns  
from xgboost import XGBClassifier  
from sklearn.model\_selection import train\_test\_split   
from sklearn.metrics import accuracy\_score  
sns.set()  
## load breast cancer datset  
df = np.loadtxt('data/diabetes.csv',delimiter=",")  
# split data into X and y  
X = df[:,0:8]  
y = df[:,8] # last column  
# split data into train and test sets  
seed = 7  
test\_size = 0.33  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  
test\_size=test\_size,random\_state=seed)  
# fit model on training data  
model = XGBClassifier(use\_label\_encoder=False)  
model.fit(X\_train, y\_train)  
  
# make predictions for test data  
predictions = model.predict(X\_test)  
  
# evaluate predictions  
accuracy = accuracy\_score(y\_test, predictions)   
print("Accuracy: %.2f%%" % (accuracy \* 100.0))

The xgboost provide a classification accuracy of 74.02%

74.02%

##### 5.2.5.1.1 XGBoost With k-Fold Cross-Validation.

Cross-validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts (e.g. k = 5 or k = 10). Each split of the data is called a fold. The algorithm is trained on k − 1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross-validation you end up with k-different performance scores that you can summarize using a mean and a standard deviation.

# Vote(Stacking) Classification  
import matplotlib.pyplot as plt  
import pandas as pd  
import numpy as np  
import seaborn as sns  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import cross\_val\_score  
from xgboost import XGBClassifier  
from sklearn.model\_selection import train\_test\_split   
from sklearn.metrics import accuracy\_score  
sns.set()  
## load breast cancer datset  
df = np.loadtxt('data/diabetes.csv',delimiter=",")  
# split data into X and y  
X = df[:,0:8]  
y = df[:,8] # last column  
  
# CV model  
model = XGBClassifier(use\_label\_encoder=False)  
kfold = KFold(n\_splits=10, random\_state=7,shuffle=True)  
results = cross\_val\_score(model, X, y, cv=kfold)  
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

The xgboost provide a classification mean accuracy of .7253

0.7253

**Activity 5.2.5.1 Gr on XGBoost Accuracy**

**Instruction**

* Be in group of 5 – 6 people
* Discuss the following question in your group, and compile all ideas and present the work to the plenary.

**Time allowed:** 5 Minutes

**Discussion Question:** Is the accuracy XGBoost with Cross-Validation improved? If / If not, explain the reason?

#### 

#### 5.2.5.2 Visualize Individual Trees of XGBoost

# Vote(Stacking) Classification  
import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
from xgboost import XGBClassifier  
from xgboost import plot\_tree  
sns.set()  
## load breast cancer datset  
df = np.loadtxt('data/diabetes.csv',delimiter=",")  
# split data into X and y  
X = df[:,0:8]  
y = df[:,8] # last column  
# fit model on training data  
model = XGBClassifier(use\_label\_encoder=False)  
model.fit(X, y)  
# plot single tree  
plot\_tree(model)  
plt.show()

#### 5.2.5.3 XGBoost Performance and Early Stopping

#### 5.2.5.4 Trained XGBoost Models