12. SUPPORT VECTOR MACHINES, DECISION TREES, RANDOM FORESTS

LEARNING OBJECTIVES

- SVM
- Decision Trees
- Random
  Forests

- Project 2 - Caravan Dataset => Tuesday 8/2
- Final Project part 2 and 3 - Design Writeup and Exploratory Data Analysis For => Tuesday 8/9

REVIEW OF LESSON 11

LAST LESSON REVIEW

- Accuracy Paradox
- Imbalanced datasets strategies
  - 3 strategies we saw?

PART I SUPPORT VECTOR MACHINES

HYPERPLANE

Classification separating an hyperplane

- 1D => point
- 2D => line
- 3D =>
  surface
- etc ...

A hyperplane in 2 dimension is defined by a line equation $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$

In **p dimensions** a hyperplane is defined by a **linear** equation

$$\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = 0 \qquad p > 0$$

LINEARLY SEPARABLE

We have n samples in p dimensions/features $X = \{x_i\} \quad i \in [1, n]$.

These observations belong to two classes $y_i \in \{-1, +1\}$

The classes are **linearly separable**: there is an hyperplane that fully separates the points according to their classes.

- All x_i such that $\beta_0 + \sum_{j=1}^{n} \beta_j x_{i,j} < 0$ belong to class -1
- All x_i' such that $\beta_0 + \sum_{j=1}^{n} \beta_j x'_{i,j} > 0$ belong to class +1

LINEAR DECISION BOUNDARY

We predict the class of a new point x

by calculating

$$f(x) = \beta_0 + \sum_{j=1}^{n} \beta_j x_j$$

- if $f(x) > 0 \Rightarrow +1$
- if $f(x) < 0 \Rightarrow -1$

A classifier that is based on a separating hyperplane has a **linear decision boundary**.

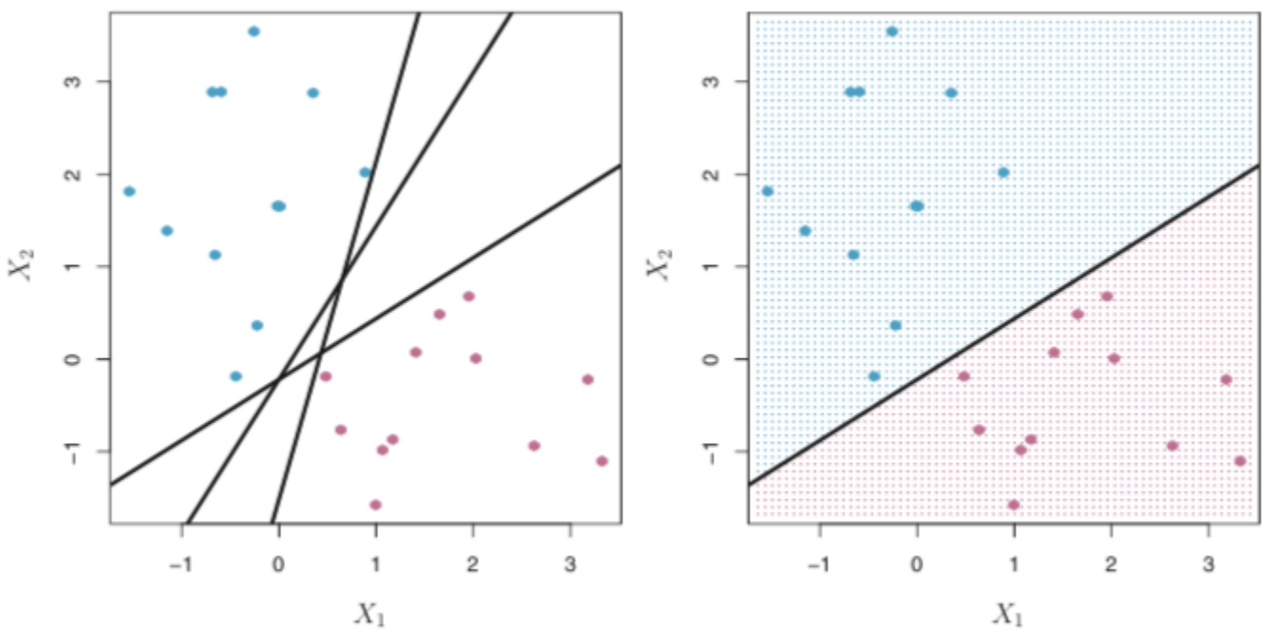
ONE LINE

Note that since y_i and $\beta_0 + \sum_{j=1}^{n} \beta_j x_{i,j}$ have the same sign:

$$y_i(\beta_0 + \sum_{j=1}^{n} \beta_j x_{i,j}) > 0$$

INFINITY OF HYPERPLANES

If our data is linearly separable then there exists an infinity of hyperplanes that can separate it



=> We need to introduce a margin to separate the classes

Fig from Introduction to Statistical Learning

MAXIMAL MARGIN CLASSIFIER

We want to find the Hyperplane that will maximise the distance to all the points.
Best separation of the classes

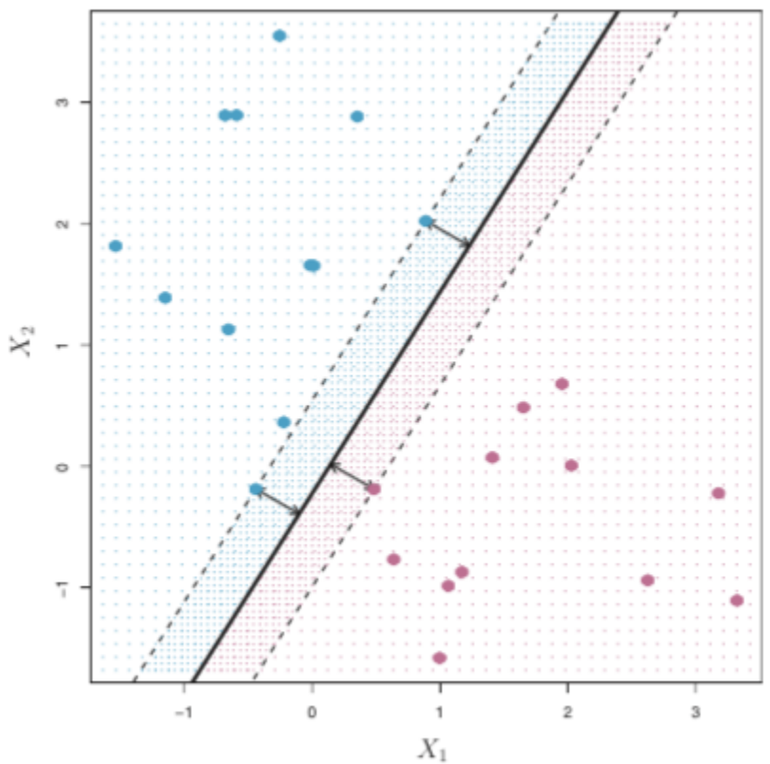


Fig from Introduction to Statistical Learning

MAXIMAL MARGIN CLASSIFIER

Fnding a simple linear boundary is equivalent to solving

$$y_i(\beta_0 + \sum_{j=1}^{n} \beta_j x_{i,j}) > 0 \quad \forall i \in [1, \ldots, n]$$

Finding **the largest margin** that separates the classes is equivalent to solving

$$\max_{\beta_i} M \quad \text{such that} \quad y_i(\beta_0 + \sum_{j=1}^{n} \beta_j x_{i,j}) > M \quad \forall i \in [1, \ldots, n]$$

with $\sum_{j=0}^{p} \beta_j^2 = 1$

SUPPORT VECTORS

- The observations that are on the margin lines are called **support vectors**
- They *support* the maximal margin hyperplane in the sense that if these points were moved slightly then the **maximal margin hyperplane** would move as well.
- The maximal margin hyperplane depends directly on the support vectors, but not on the other observations. A movement to any of the other observations would not affect the separating hyperplane, provided that the observation's movement does not cause it to cross the boundary set by the margin.

A change in the support vector impacts the margin a lot => **over fitting**

SUPPORT VECTOR CLASSIFIER

What if we allow some points to be either wrongly classified or at least within the margin boundaries?

NON LINEARLY SEPARABLE

This is the case when the classes are not separable, we still want the best margin possible. But some points will be on the wrong side

Add a tuning parameter C that dictates the *severity of the violation* of the margin.

The bigger C is the more points are **within the margin** or **misclassified**. C is some budget for violation of the margin which is determined during cross validation.

=> Even in the case of linearly separable classes adding some flexibility to the margin will make the classfier more robust, more flexible (less overfitting)

SUPPORT VECTOR CLASSIFIER

C is a non negative tuning parameter

$$\max_{\beta_i, \epsilon_i} M \quad \text{such that} \quad y_i(\beta_0 + \sum_{j=1}^{n} \beta_j x_{i,j}) > M - \epsilon_i \quad \forall i \in [1, \ldots, n]$$

with

- $\sum_j \beta_j^2 = 1$
- $\epsilon_i > 0$
- $\sum_i \epsilon_i \leq C$

LAB SVC

L12 N1 Support Vector Classifier - Linear Case.py

SUPPORT VECTOR MACHINE

What if the data is not even close to be linearly separable?

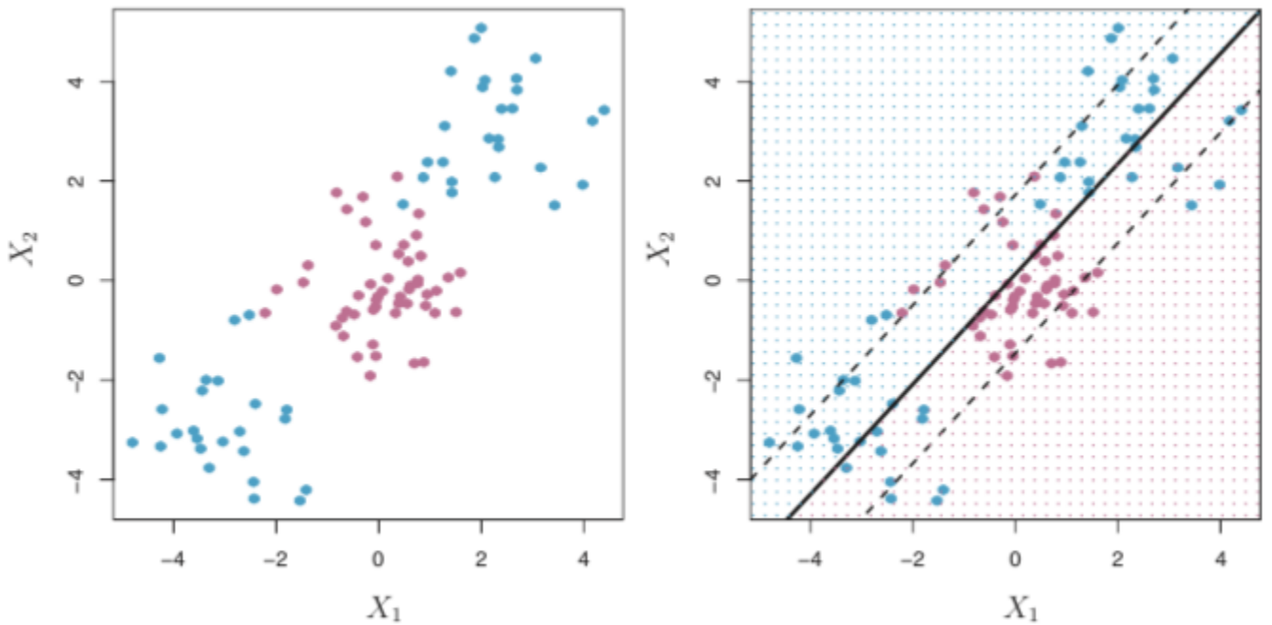


Fig from Introduction to Statistical Learning

INTRODUCING KERNELS

The optimization equation for the linear support vector classifier can be rewritten as such

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i K(x, x_i)$$

with $K(x, x_i) = \langle x, x_i \rangle$ the vector dot product.

But we could use a different Kernel function

Note:This involves $p * \frac{n(n-1)}{2}$ multiplications! However only the support vectors are useful. So we can limit the sum to S support vectors.

KERNELS

Here are some classic Kernel functions

- Linear Kernel $K(x, x_i) = \langle x, x_i \rangle$
- Polynomial Kernel (d): $K(x, x_i) = (1 + \sum_{j=1}^{p} x_j x_{i,j})^d$
- Radial Kernel $K(x, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x_j - x_{i,j})^2)$

SVM LAB

L12 N2 Support Vector Classifier - Non-Linear Case.py

PART II DECISION TREE

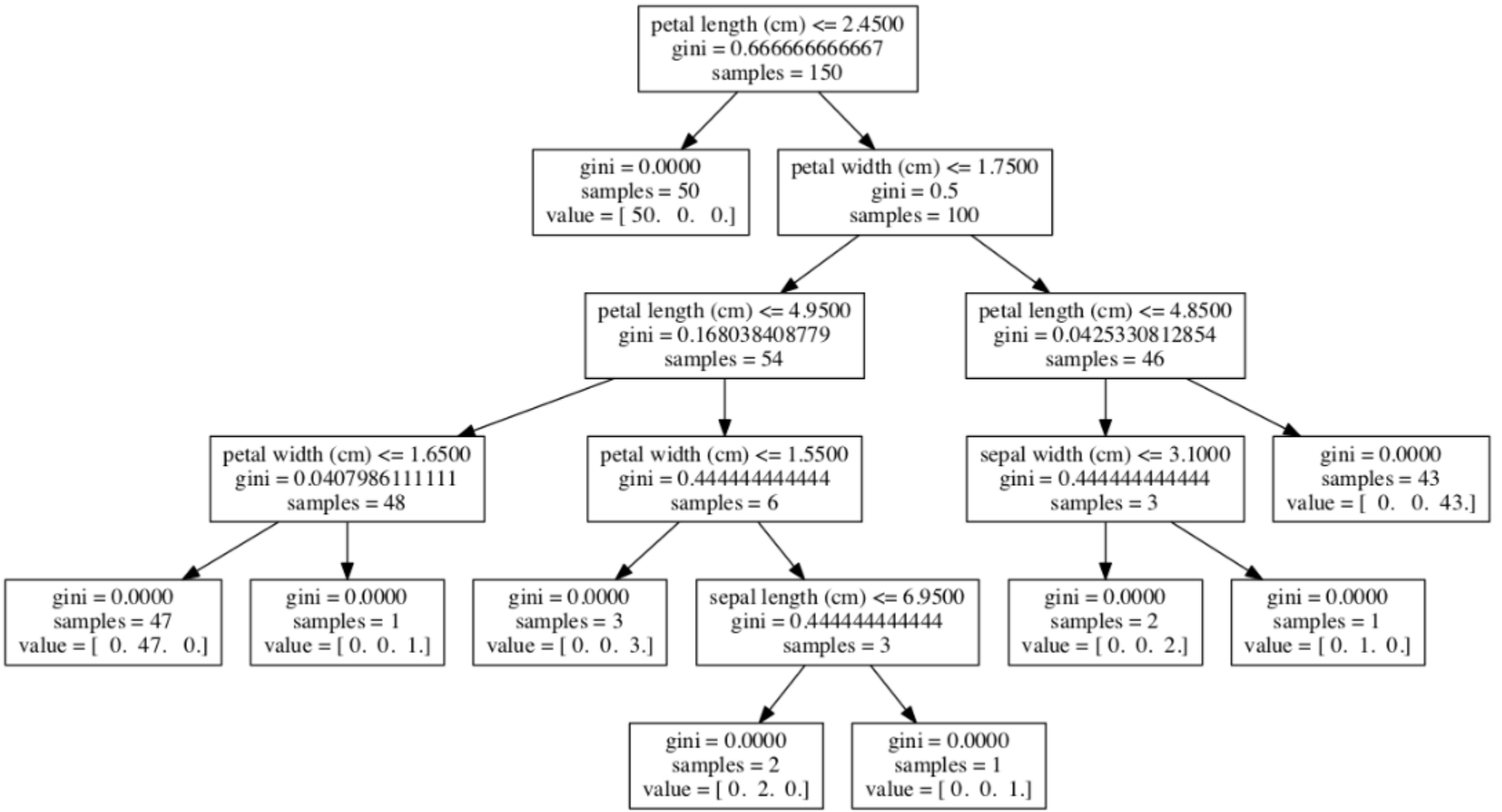
DECISION TREES

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Rule based models

```
if x < a
    if y < c
        ...
    else
        ...
elif x > a & a < b
    ...
else
    if z < d
        ...
    else
        0
```

IRIS DATASET



DECISION TREES

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. (missing values, scaling, dummy variables, ...)
- Low cost for prediction $O(log(n))$ with n number of data points used to train the tree.
- Can handle both numerical and categorical data.
- Uses a white box model. An observed situation can simply be explained by boolean logic.
- Possible to validate a model using statistical tests.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

DECISION TREES

- high overfitting for over-complex trees that do not generalise the data well.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated.
- no globally optimal decision tree

LAB: CONTROLLING THE TREE

Set these params to control the tree complexity

- max_depth
- min_samples_split
- min_samples_leaf
- max_features

Lab: Simple Decision tree

Weak Learner: What if we forcefully prune the tree?

CLASSIFICATION VS REGRESSION

Different Metric

- MSE
- Gini

BOOTSTRAP AGGREGATION AKA BAGGING

We used Boostrapping to estimate the mean of a sample.

Sampling with replacement.

The idea is the same with trees or any other classifier.

BAGGING FOR TREES

- Generate B different bootstrapped training data sets.
- Train a new tree on each training set

The predictions of all the trees are averaged

=> significantly reduces over fitting

OUT OF BAG - OOB

When boostrapping, in each experiment will use only approx. 2/3rd of the available samples.

Which leaves 1/3rd that we can use to estimate the validation error of each tree.

This is called OOB Out of Bag error.

It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

FEATURE IMPORTANCE

- When the *max_features* < *total number of features*.

  => Some features are left out of the splitting decision in each node.

- Relative Feature importance can be deduced from the delta in MSE associated to the features included vs left out.

BAGGING IN SCIKIT

Use the Bagging Classifier

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

BAGGING IN SCIKIT

Very flexible

```
    bagging = BaggingClassifier(DecisionTreeClassifier(),  boostrap = True)
```

class sklearn.ensemble.BaggingClassifier

- base_estimator: The base estimator, decision tree by default
- n_estimators: How many estimators will be ensembled
- max_samples: The number of samples to draw from X to train each base estimator
- max_features: The number of features to draw from X to train each base estimator
- bootstrap: True / False
- bootstrap_features: True / False
- oob_score: True / False

BAGGING IN SCIKIT

Lab - bagging with Tree - regression

RANDOM FORESTS

Extension of the bootstrapping to features

- In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
- In addition, when splitting a node during the construction of the tree, the split that is picked is the best split among **a random subset of the features**.

=> The bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

(see RandomForestClassifier and RandomForestRegressor classes),

RANDOM FOREST LAB

Random Forests Lab

RECAP

- Support Vector Classifiers
- Decision Trees
- Bagging
- Random Forests

- Project 2 for next Tuesday
- Final Project part 2 and 3 for 8/9

QUESTIONS

QUESTIONS?