Supervised learning: Classification

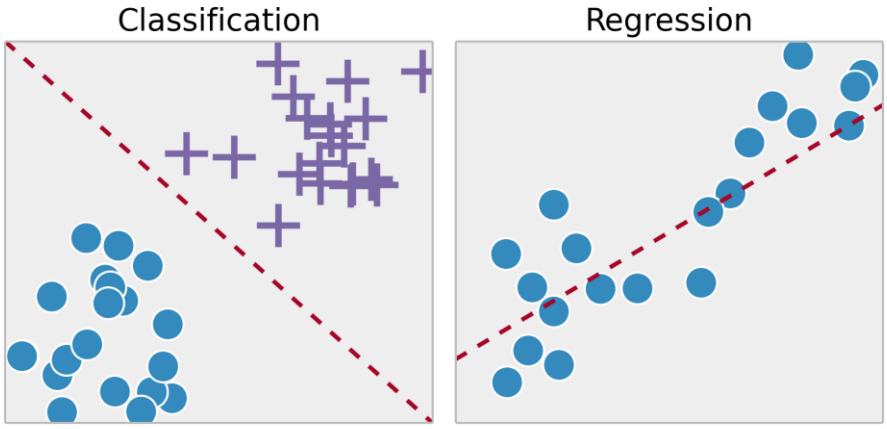
1 Outline

- Classification overview
- Evaluation on classification models/results
- Basic classification models
- Ensemble models

2 Classification Overview

2.1 Statistical Classification

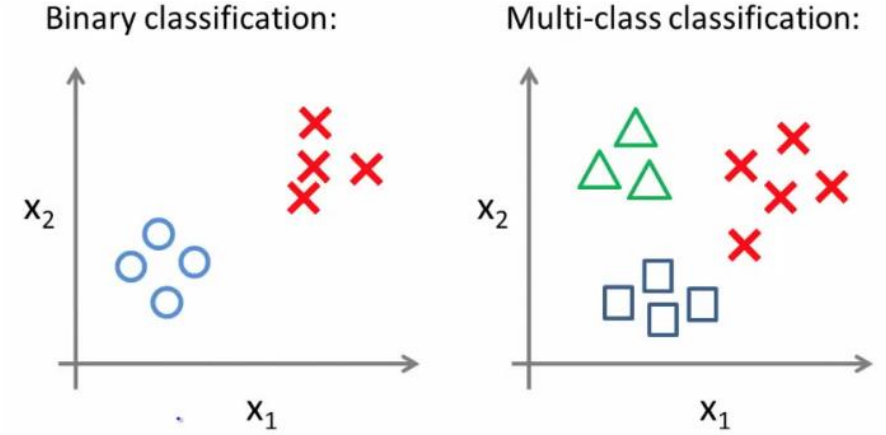
Identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known.



We have our data on the x-axis and some objective function on y-axis. We try to predict with the function and give the data (dots) best fit line which is the red line we gave in the plot. Regression is a very powerful tool, but it doesn' t apply to every type of problem. So regression problems work when we are mapping our data to some numeric value answer. Classification is where we are trying to separate two data sets. In this case, we have the pluses and dots to representative two classes. We want to find a separation between those.

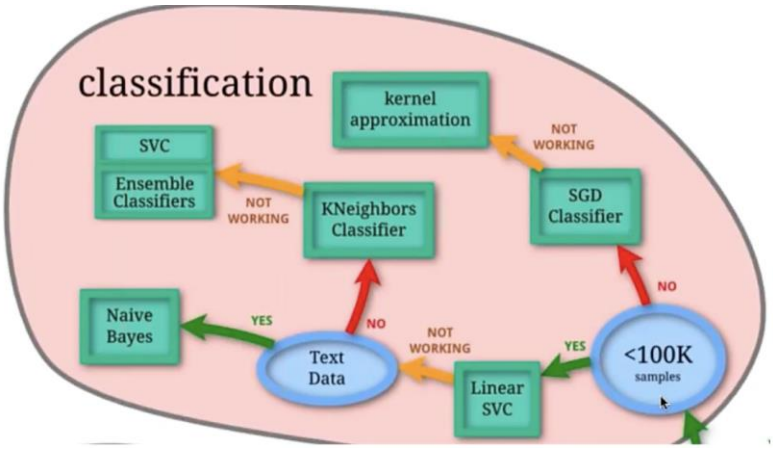
2.2 Classification Types (binary vs. multi-class)

Multi-class could be treated as multiple binary classes, focus on binary first . In machine learning, **binary classification** is the task of classifying the elements of a given set into two groups on the basis of a classification rule. **Multiclass classification** is the problem of classifying instances into one of the more than
binary classification. Multi-class could be treated as multiple binary classes, focus on binary first. In the Binary classification graph, we set x and o as true and false. In fact, we might face larger problems like we have three classes in multi-class classification graph.



2.3 Classification Algorithms

- Algorithm have their own pros and cons. Among them the main one is: Complexity vs. Generality
- Scikit-learn cheatsheet：
  http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
  These processes would be good algorithms to learn about model evaluation. This bubble is about the classification section and this is the recommended decision tree that provides for how we decide what kind of classifier to use.



3 Evaluation on classification models/results

3.1 Model Evaluation – General

In the last section, we talked about diagnostics and model measures, and we covered a lot of works. In here, we will make a quick summary at a higher level. In this section, we should remember that we want to know how important the cross validation is if the data is limited. To know which one works better, for a specific problem:

Hold-out:

• Randomly divide dataset into: 1. training 2. validation 3. Test Cross-validation:

• If data is limited.

3.2 Model Evaluation – Scikit Learn Functions

There are some basic functions of model evaluation that we should know:

`metrics.accuracy_score`(y_true, y_pred[, ...])	Accuracy classification score.
`metrics.auc`(x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
`metrics.average_precision_score`(y_true, y_score)	Compute average precision (AP) from prediction scores
`metrics.brier_score_loss`(y_true, y_prob[, ...])	Compute the Brier score.
`metrics.classification_report`(y_true, y_pred)	Build a text report showing the main classification metrics
`metrics.confusion_matrix`(y_true, y_pred[, ...])	Compute confusion matrix to evaluate the accuracy of a classification
`metrics.f1_score`(y_true, y_pred[, labels, ...])	Compute the F1 score, also known as balanced F-score or F-measure
`metrics.fbeta_score`(y_true, y_pred, beta[, ...])	Compute the F-beta score
`metrics.hamming_loss`(y_true, y_pred[, classes])	Compute the average Hamming loss.
`metrics.hinge_loss`(y_true, pred_decision[, ...])	Average hinge loss (non-regularized)
`metrics.jaccard_similarity_score`(y_true, y_pred)	Jaccard similarity coefficient score
`metrics.log_loss`(y_true, y_pred[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
`metrics.matthews_corrcoef`(y_true, y_pred)	Compute the Matthews correlation coefficient (MCC) for binary classes
`metrics.precision_recall_curve`(y_true, ...)	Compute precision-recall pairs for different probability thresholds
`metrics.precision_recall_fscore_support`(...)	Compute precision, recall, F-measure and support for each class
`metrics.precision_score`(y_true, y_pred[, ...])	Compute the precision
`metrics.recall_score`(y_true, y_pred[, ...])	Compute the recall
`metrics.roc_auc_score`(y_true, y_score[, ...])	Compute Area Under the Curve (AUC) from prediction scores
`metrics.roc_curve`(y_true, y_score[, ...])	Compute Receiver operating characteristic (ROC)
`metrics.zero_one_loss`(y_true, y_pred[, ...])	Zero-one classification loss.
`metrics.brier_score_loss`(y_true, y_prob[, ...])	Compute the Brier score.

3.3 Model Evaluation - Confusion Matrix

A **confusion matrix** is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology may be confusing and need special efforts to figure them out.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	b/(b+d)		

How many selected items are relevant?

Precision = $\frac{\text{green}}{\text{green + red}}$

Spam email: 1000 emails, 10 spam; a **spam filter** find out 15 "spams", 3 out of 15 are real spams

Q: what is the sensitivity (recall), precision, and total accuracy?

Answer:

	Target(Spam)	Target(Non-spam)
Model(Spam)	3	12
Model(Non-spam)	7	978

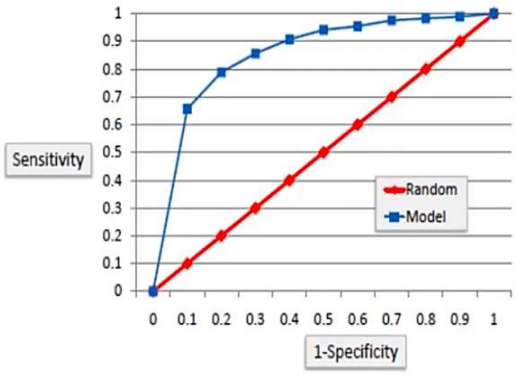
Sensitivity = 3 / (3 + 7) = 0.3; Specificity = 978/(978+12); Precision = 3/(3+12) = 0.2; Accuracy=(978+3)/1000 = 98.1%

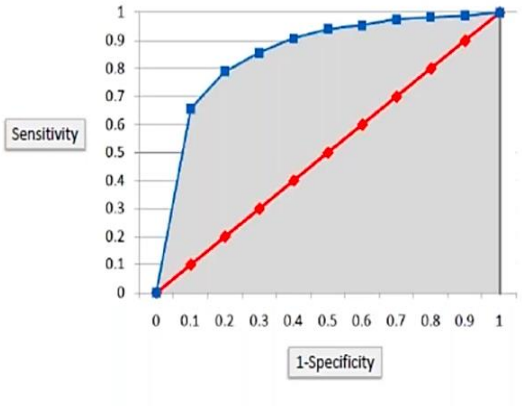
3.4 Model Evaluation - ROC Curve

The **ROC curve** is created by plotting the **true positive rate (TPR)** against the **false positive rate (FPR)** at various threshold settings. TThis type of graph is called a Receiver Operating Characteristic curve (or ROC curve.) It is a plot of the true positive rate against the false positive rate for the different possible cutpoints of a diagnostic test.

An ROC curve demonstrates several things:

- It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.





3.5 Examples in Python

```
In [ ]: # accuracy_score, precision_score, recall_score, f1_score,
        # confusion_matrix, classification_report
```

```
In [7]: from sklearn import metrics
```

```
In [2]: y_true = [0, 1, 0, 1, 1, 1, 0, 0] # raw data
        y_pred = [0, 1, 1, 1, 1, 1, 0, 1] # final prediction
        metrics.classification.confusion_matrix(y_true, y_pred)
```

```
Out[2]: array([[2, 2],
               [0, 4]])
```

```
In [13]: metrics.classification.precision_score(y_true, y_pred, pos_label=0)
```

```
Out[13]: 1.0
```

```
In [3]: metrics.classification.precision_score(y_true, y_pred, pos_label=1)
```

```
Out[3]: 0.6666666666666663
```

```
In [4]: metrics.classification.recall_score(y_true, y_pred, pos_label=0)
```

```
Out[4]: 0.5
```

```
In [15]: metrics.classification.recall_score(y_true, y_pred, pos_label=1)
```

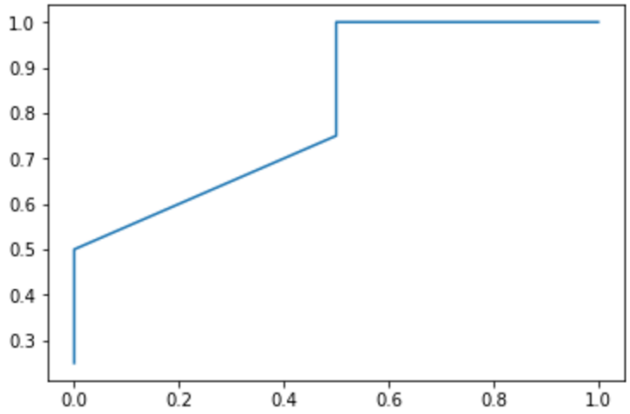
```
Out[15]: 1.0
```

```
In [10]: # ROC & AUC
         y_true = [0, 1, 0, 1, 1, 1, 0, 0] # raw data
         y_score = [0.2, 0.7, 0.6, 0.6, 0.5, 0.9, 0.4, 0.6] # problistic results
         # y_pred = [0, 1, 1, 1, 1, 1, 0, 1] # final prediction

In [11]: x, y, thed = metrics.roc_curve(y_true, y_score, pos_label=1)

In [12]: %matplotlib inline
         plt.plot(x, y)

Out[12]: [<matplotlib.lines.Line2D at 0x1197b09b0>]
```



4 Basic Classification Models

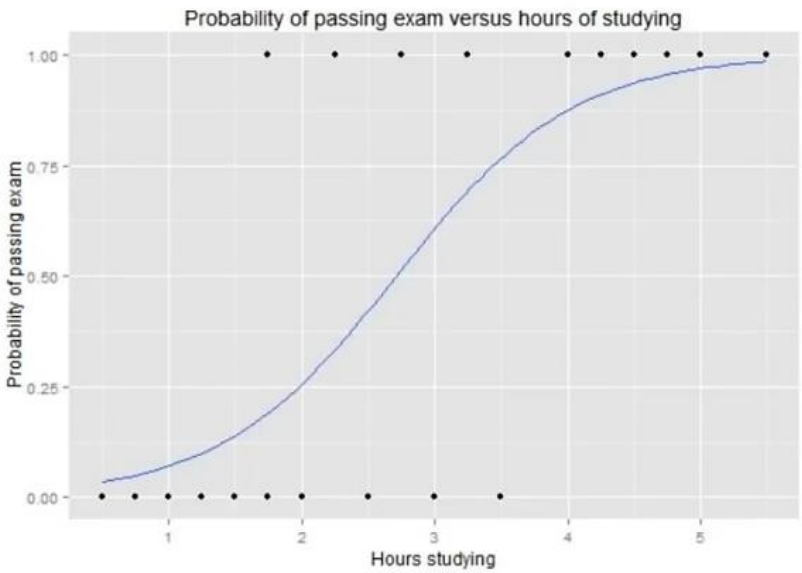
We will talk about some basic classification models in this part. The main basic classification models are Logistic Regression, Decision Tree, and Support Vector Machine (SVM).

4.1 Model: Logistic Regression

The prediction of linear regression is numerical, but the logistic regression can make a prediction of a class (such as true or false, red or blue). It is an extension of linear regression allowing for binary classification. Non-linear transformation over linear combination (the logistic function can now be written as):

$$F(x) = \frac{1}{1 + e^{-(\beta 0 + \beta 1 x)}}$$

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is useful because it can take any real input, whereas the output always takes values between zero and one and hence can be interpretable as a probability.



4.2 Interpreting the coefficient

Log odds ratio:

$$P(x) = F(x) = \frac{1}{1 + \exp[-(\beta_0 + \beta_1 x)]}$$

$$\frac{1}{P(x)} = 1 + \exp[-(\beta_0 + \beta_1 x)]$$

$$\frac{1 - P(x)}{P(x)} = \exp[-(\beta_0 + \beta_1 x)]$$

$$(\beta_0 + \beta_1 x) = \log \frac{P(x)}{1 - P(x)}$$

$$odds = \frac{P(x)}{1 - P(x)}$$

4.2.1 Examples in Python-– Logistic Regression

In [45]:
```
from sklearn import linear_model
```

In [46]:
```
linear_model.LogisticRegression?
```

In [47]:
```
from sklearn import datasets
```

In [55]:
```
X, Y = datasets.make_blobs(n_samples=100, n_features=2,
                           centers=2, cluster_std=2, random_state=3)
```

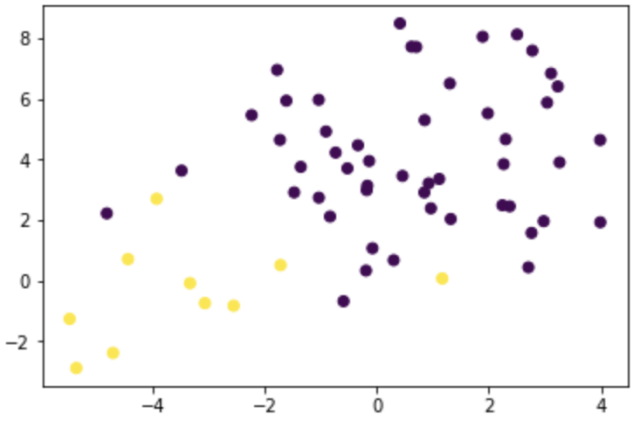
In [62]:
```
# down-sampling
m = 10
X0 = X[Y==0]
X1 = X[Y==1][:m]
print(X0.shape, X1.shape)

Xnew = np.vstack([X0, X1])
Ynew = [0 if i < X0.shape[0] else 1 for i in range(50+m)]
```

```
(50, 2) (10, 2)
```

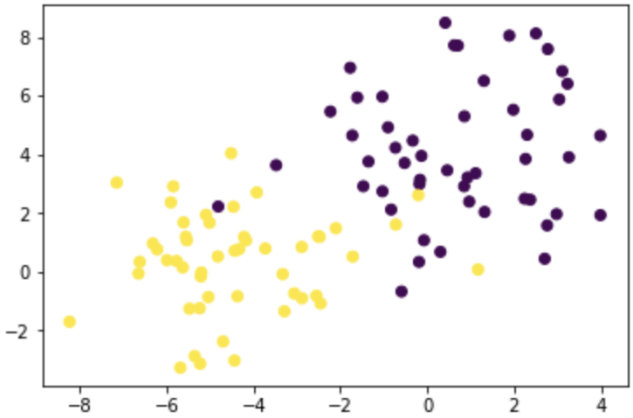
In [63]:
```
plt.scatter(Xnew[:,0], Xnew[:,1], c=Ynew)
```

Out[63]:
```
<matplotlib.collections.PathCollection at 0x11dd85fd0>
```



In [60]:
```
plt.scatter(X[:,0], X[:,1], c=Y)
```

Out[60]:
```
<matplotlib.collections.PathCollection at 0x1126bdba8>
```



In [64]:
```
lm_lr = linear_model.LogisticRegression()
```

In [65]:
```
lm_lr.fit(X, Y)
```

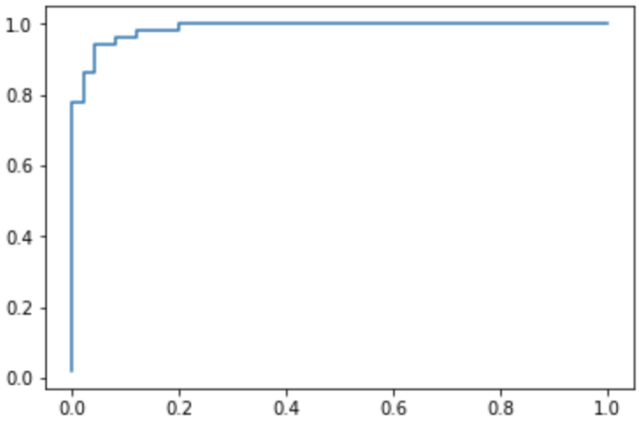
Out[65]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001
,
          verbose=0, warm_start=False)
```

In [67]:
```
metrics.confusion_matrix(Y, lm_lr.predict(X))
```

Out[67]:
```
array([[48,  2],
       [ 3, 47]])
```

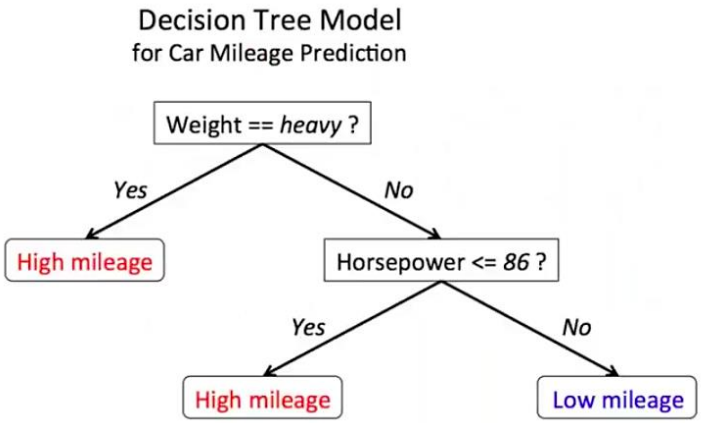
In [76]:
```
x_v, y_v, thred = metrics.roc_curve(Y, lm_lr.predict_proba(X)[:,1])
%matplotlib inline
plt.plot(x_v, y_v)
```

Out[76]:
```
[<matplotlib.lines.Line2D at 0x11e01b518>]
```



4.3 Model: Decision Tree

This part introduces the decision tree classifier, which is a simple yet widely used classification technique. Decision Tree is very popular in recent because it is especially good for dealing with non-linear data.



A **decision tree** is a machine learning algorithm that partitions the data into subsets. The partitioning process starts with a binary split and continues until no further splits can be made.

4.3.1 How Decision Tree Works?

All decision tree models are heuristic (may not be globally optimal)
Three steps:

- Which feature to split?
- How to split on the selected features?
- When to stop?

4.3.2 Three metric for split

Classification:
Gin impurity: $\boldsymbol{H(X_m) = \sum_k P_{mk}(1 - P_{mk}),\ P_{mk} = 1/N_m \sum_{x_i \in R_m} I(y_i = k)}$
Information gain: $\boldsymbol{H(X_m) = \sum_k P_{mk} \log(P_{mk})}$
Regression
Mean Square Error: $\boldsymbol{C_m = \frac{1}{N_m} \sum_{i \in N_m} y_i,\ H(X_m) = \frac{1}{N_m} \sum_{i \in N_m} (y_i - C_m)^2}$

4.3.3 Gini Impurity

Feature Value (X1)	Class (Y)
1	A
2	B
3	A
4	B
5	B
6	A

If Split at 3.5, there will be two nodes.
Before split:
P(A) = 0.50, P(B) = 0.50
After split:
Node 1: A, B, A. P(A) = 0.67, P(B) = 0.33
Node 2: B, B, A. P(A) = 0.33, P(B) = 0.67

Gini impurity (gain)
Before: (all nodes) P(A) * (1-P(A)) + P(B) * (1-P(B)) = 0.5*0.5+0.5*0.5=0.5
After: (% node 1) P(A) * (1-P(A)) + P(B) * (1-P(B)) + (% node 2) P(A) * (1-P(A)) + P(B) * (1-P(B)) =
0.5 * (0.67 * 0.33 + 0.33 * 0.67) + 0.5 * (0.33 * 0.67 + 0.67 * 0.33) = 0.4422

4.3.4 Information Gain

Feature Value (X1)	Class (Y)
1	A
2	B
3	A
4	B
5	B
6	A

If split at 3.5, there will be two nodes
Before split:
P(A) = 0.50, P(B) = 0.50
After split:
Node 1: A, B, A. P(A) = 0.67, P(B) = 0.33
Node 2: B, B, A. P(A) = 0.33, P(B) = 0.67

Before: -P(A) * log(P(A)) - P(B) * log(P(B)) = - 0.5 * log(0.5) – 0.5 * log(0.5) = 0.693
After: (% node 1) -P(A) * log(P(A)) - P(B) * log(P(B)) + (% node 2) -P(A) * log(P(A)) - P(B) * log(P(B)) =
0.5 * (-0.67 * log(0.67) - 0.33 * log(0.33) -0.67 * log(0.67) - 0.33 * log(0.33)) = 0.634

4.3.5 Use Gini (Example)

Feature Value (X1)	Class (Y)
1	A
2	B
3	A
4	B
5	B
6	A

Split can happen on: 1.5, 2.5, 3.5, 4.5, 5.5.

Before split: P(A) = 0.50, P(B) = 0.50

	Node 1&2		Gini 1&2	Total Gain
1.5	PA=1.0,PB =0.0, N=1	PA=0.4,PB =0.6, N=5	1/6 * 0 + 5/6 * 0.48 = 0.40	0.5 – 0.4 = 0.1
2.5	PA=0.5,PB =0.5, N=2	PA=0.5,PB =0.5, N=4	2/6 * 0.5 + 4/6 * 0.5 = 0.5	0.5 – 0.5 = 0
3.5	PA=2/3,PB =1/3, N=3	PA=1/3,PB =2/3, N=3	3/6 * 0.44 + 3/6 * 0.44 = 0.44	0.5 - 0.44 = 0.06
4.5	PA=0.5,PB =0.5, N=4	PA=0.5,PB =0.5, N=2	4/6 * 0.5 + 2/6 * 0.5 = 0.5	0.5 – 0.5 = 0
5.5	PA=0.4,PB =0.6, N=5	PA=1.0,PB =0.0, N=1	5/6 * 0.48 + 1/6 *0 = 0.40	0.5 – 0.4 = 0.1

5. Example in Python

Decision boundary

In [ ]:
```
# how to plot a decision boudnary?

lm_lr = linear_model.LinearRegression()
Ypred = lm_lr.predict(X)
```

In [65]:
```
plt.contourf?
```

Probability calibration

In [69]:
```
centers = [(-5, -5), (0, 0), (5, 5)]
X, y = datasets.make_blobs(n_samples=100, n_features=2, cluster_std=1.0,
                  centers=2, shuffle=False, random_state=42)
```

In [74]:
```
dtc = tree.DecisionTreeClassifier(max_depth=5)
```

In [75]:
```
dtc.fit(X, Y)
```

Out[75]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

In [77]:
```
from sklearn import calibration
```

In [81]:
```
ir = calibration.IsotonicRegression?
```

In [85]:
```
from sklearn import pipeline
```

In [ ]:
```
pipeline.Pipeline([('lgoistic regression', lm_lr), ('calibration', lr)])
```

In [83]:
```
ir.fit(dtc.predict_proba(X)[:,1], Y)
```

Out[83]:
```
IsotonicRegression(increasing=True, out_of_bounds='nan', y_max=None,
          y_min=None)
```

In [84]:
```
ir.predict(dtc.predict_proba(X)[:,1])
```

Out[84]:
```
array([ 1.        ,  0.46511628,  0.5       ,  0.        ,  0.46511628,
        0.        ,  0.        ,  1.        ,  0.46511628,  0.        ,
        1.        ,  1.        ,  1.        ,  1.        ,  0.        ,
        0.46511628,  0.46511628,  1.        ,  0.46511628,  0.        ,
        0.        ,  0.5       ,  0.46511628,  0.46511628,  0.        ,
        0.46511628,  1.        ,  0.46511628,  1.        ,  1.        ,
        0.46511628,  0.        ,  0.46511628,  0.46511628,  1.        ,
        1.        ,  0.        ,  0.        ,  0.        ,  0.46511628,
        1.        ,  0.46511628,  0.        ,  0.        ,  1.        ,
        1.        ,  0.46511628,  0.        ,  1.        ,  1.        ,
        0.46511628,  1.        ,  1.        ,  1.        ,  0.46511628,
        0.46511628,  0.        ,  1.        ,  0.        ,  0.        ,
        0.46511628,  0.        ,  0.        ,  0.46511628,  0.46511628,
        0.46511628,  1.        ,  0.        ,  0.46511628,  0.46511628,
        0.46511628,  1.        ,  0.46511628,  0.46511628,  0.46511628,
        0.        ,  1.        ,  0.46511628,  0.46511628,  1.        ,
        0.46511628,  0.46511628,  0.        ,  0.        ,  0.46511628,
        0.46511628,  0.46511628,  0.46511628,  0.        ,  0.46511628,
        0.46511628,  1.        ,  1.        ,  0.46511628,  1.        ,
        0.46511628,  0.46511628,  0.        ,  0.46511628,  1.        ])
```

Ensemble models

Ensemble modeling is the process of running two or more related but different analytical models and then synthesizing the results into a single score in order to improve the accuracy of predictive analytics and data mining applications. The main ways of ensemble models are Averaging and Boosting.

Averaging

Build several estimators independently and then to average their predictions.

Bagging

- "Bagging" = bootstrap aggregation
  Learn many classifiers, each classifier is learned with only a part of the data
  Combine through model averaging
- Bagging (stands for Bootstrap Aggregation) is the way decrease the variance of your prediction by generating additional data for training from your original dataset using combinations with repetitions to produce multisets of the same cardinality/size as your original data. By increasing the size of your training set you can't improve the model predictive force, but just decrease the variance, narrowly tuning the prediction to expected outcome.

Random Forest

- There is a good example for Ensemble Model deal with a classification problem: Random forest Classifier. Random Forests have a lot of classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. When we let the trees “vote”, the voting like the weighted combinations of predictors. Then the forest offer a best classification depend on the voting number, the process like a “committee” decisions.

- **More information:** https://www.quora.com/How-does-randomization-in-a-random-forest-work?redirected_qid=212859

Boosting

Boosting is a two-step approach, where one first uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.

Adaboost

Adaptively change point weight. At each step of iteration, we apply the base algorithm to the training set and increase the weights of the incorrectly.

Gradient Boosting

- Negative functional gradient. Similar as Adaboots, we redo the incorrectly parts to the original function, and get a function gradient, then we get the new parameter.
- More information: https://www.analyticsvidhya.com/blog/2015/09/complete-guide-boosting-methods/