K-Nearest Neighbors

Tuesday, 23 October 2018 10:39

1 k-Nearest Neighbors

Problem 1: Consider a dataset with 3 classes $\mathcal{C} = \{A, B, C\}$, with the following class distribution $N_A = 42, N_B = 67, N_C = 27$. We use unweighted k-NN classifier, and set k to be equal to the number of data points, i.e. $k = N_A + N_B + N_C =: N$. What can we say about the prediction for a new point x_{new}?

A) x_{new} will be classified as class A

B) x_{new} will be classified as class B

C) x_{new} will be classified as class C

D) We don't have enough information to answer the question

How about if we use the weighted (by distance) version of k-Nearest Neighbors?

It will be classified as class B. When k is equal to the number of data points, the neighborhood of a new point contains all points in the training set regardless of their distance. The majority class in the neighborhood is thus equal to the majority class in the dataset.

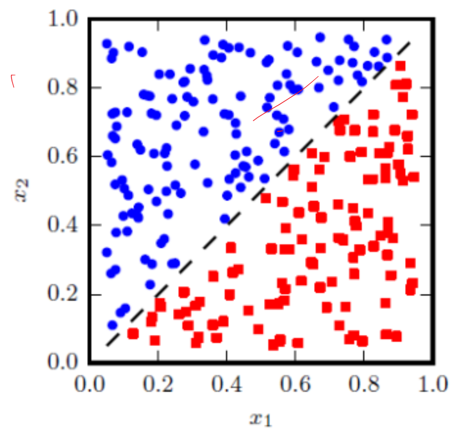
For the distance weighted variant we don't have enough information to answer the question, since the *weighted* distribution depends on the distances.

Decision trees

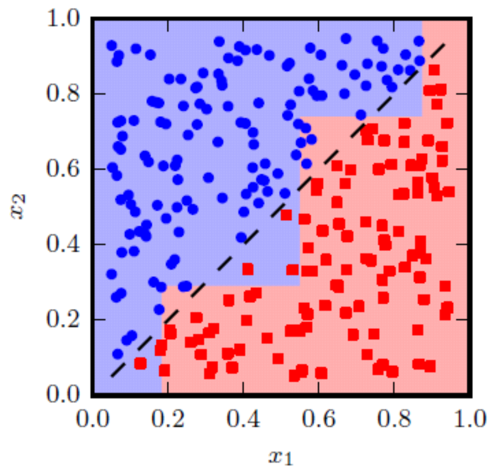
Tuesday, 23 October 2018 10:39

2 Decision Trees

Problem 2: The plot below shows data of two classes that can easily be separated by a single (diagonal) line. Does there exist a decision tree of depth 1 that classifies this dataset with 100% accuracy?



False, the feature test in a node can only use a single feature to split the training data. This leads to axis-parallel decision boundaries. Below you see the decision boundaries for a tree of depth 3. It classifies the dataset with 92.8% accuracy.



Measuring classification performance

Tuesday, 23 October 2018 10:39

How can we (quantitatively) assess the performance of a classification algorithm?

Assume that we have developed some ML algorithm and apply it the following simple binary classification problem:

Predictions of the ML algorithm

True label	Predicted label	Correct?
Cat	Cat	✓
Cat	Cat	✓
Cat	Cat	✓
Cat	Dog	✗
Cat	Dog	✗
Dog	Dog	✓
Dog	Dog	✓
Dog	Dog	✓
Dog	Cat	✗
Dog	Cat	✗

The most simple metric we can compute is **Accuracy**

$$\text{Accuracy} = \frac{\#\text{ correct preds.}}{\#\text{ of samples}}$$

For our case

$$\text{Accuracy} = \frac{6}{10} = 60\%$$

There exist scenarios, where accuracy is not the optimal metric.

Consider the following scenario: we design an algorithm that predicts whether a patient has cancer based on some information from the health records.

- Most people don't have cancer (most samples have class y=0). This means that we have a very **unbalanced class distribution** and a "simple" algorithm that always predicts "no cancer" (y_pred = 0) will have a very good accuracy (i.e. it will only make mistakes for samples where y = 1, but those are very rare). However, this is a completely useless algorithm for our problem!
- Not all mistakes (= classification errors) are equally bad. If we predict "cancer" for a healthy patient, this is not too bad, since we can perform other tests and figure out what's going on. If, however, we predict "no cancer" for a patient who is in fact ill, this is much worse!

A better way to visualize the predictions and mistakes that an algorithm makes is using a **confusion table.**

		Actual	
		Cancer	No cancer
Predicted	Cancer	30	10 I
	No cancer	10 II	950

For our case, mistakes II are much worse than mistakes I. (but it could be different for other use cases)

In general, entries of the confusion table have the following names:

		Actual	
		y=1	y=0
Predicted	y=1	TP	FP
	y=0	FN	TN

- TP: True positives
- TN: True negatives

} correct predictions

- FP: False positives (also known as Type 1 errors)
- FN: False negatives (also known as Type 2 errors)

} wrong predictions

While a confusion table provides a good insight into the predictive performance of the algorithm, it's not well suited for comparing two different ML algorithms. (For each model we have a table with 4 numbers, how can we tell which model is better for our use case?)

There are multiple ways that we can aggregate the results from the confusion table:

Accuracy:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

Same as what we used above.

Precision (positive predictive value):

$$PREC = \frac{TP}{TP + FP}$$

High precision means "If our model predicts the positive class, then it's very likely that the class is actually positive"

Use case when high precision is important - fraud detection in e-commerce.

If we classify a user as "fraudster", we want to be very sure that they are actually a fraudster. Otherwise, banning honest users will have a negative impact on the platform's reputation.

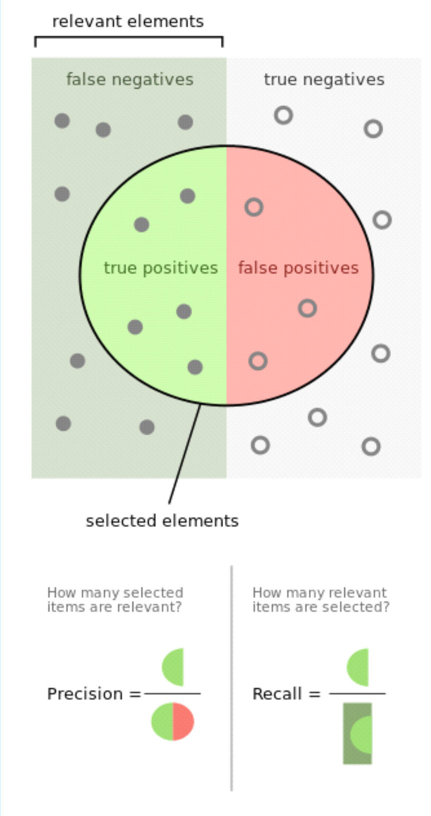
Recall (sensitivity, true positive rate):

$$REC = \frac{TP}{TP + FN}$$

High recall means "Our model is able to detect most of the positive samples"

Use case when high recall is important - cancer detection (as above).

We want to make sure that our model (test) doesn't miss any patients who are actually ill, and doesn't mistakenly classify them as healthy.



Most of the time there is a **tradeoff** between precision and recall: increasing one most often leads to decreasing the other. You should design your model (prepare the data, etc.) such that you find the correct precision-recall balance that is needed for your application.

Specifity (true negative rate):

$$TNR = \frac{TN}{FP + TN}$$

False Negative Rate (miss rate):

$$FNR = \frac{FN}{TP + FN}$$

False Positive Rate (fall out):

$$FPR = \frac{FP}{FP + TN}$$

Other metrics can also be derived from the confusion table, but they do not come up in practical applications that often.

F1 Score (harmonic mean of Recall and Precision):

$$F1 = \frac{2 * PREC * REC}{PREC + REC}$$

If we want to compare two ML models, and take both precision and recall into account, a good choice is to use the **F1-score**. It is often a good alternative to accuracy (especially, when dealing with unbalanced classes).