Lecture 4. Decision tree & Overfitting

COMP 551 Applied machine learning

Yue Li
Assistant Professor
School of Computer Science
McGill University

September 15, 2022

Outline

Learning objectives

Understanding the following concepts

- Precision-recall
- Decision tree
- Overfitting

Outline

Sensitivity/Recall, Specificity, and Precision

Sensitivity or Recall: Proportion of true positive example among ALL positive (P)

$$Sensitivity = Recall = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{1}$$

Specificity: Proportion of true negative among ALL negative (N)

$$Specificity = \frac{TN}{FP + TN} = \frac{TN}{N} \tag{2}$$

False positive rate: 1 - Specificity = $\frac{FP}{N}$

Precision: Proportion of true positive example among **the predicted positive** (PP)

$$Precision = \frac{TP}{PP} \tag{3}$$

F1-score: F1 = 2 $\times$ (precision $\times$ recall) / (precision + recall)

Precision is very important in many circumstances, e.g.,

- We can only afford testing 5 drugs among 100 predicted drugs
- We can admit a small number of high-risk patients among all patients

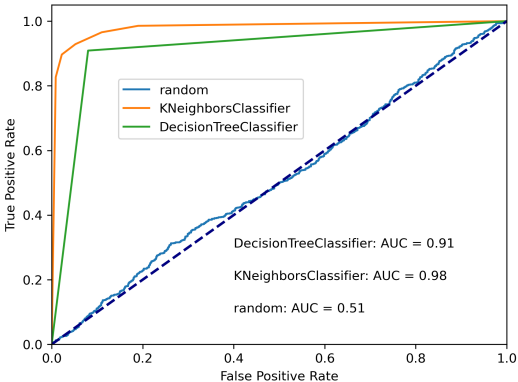
Constructing ROC and PR curve by varying the thresholds

TPR	FPR	Threshold
0.0	0.000000	1.999820
0.0	0.001111	0.999820
0.0	0.020000	0.984463
...	...	...
0.3	0.067778	0.936420
0.3	0.090000	0.918972
0.4	0.090000	0.918953
0.4	0.291111	0.719385
0.5	0.291111	0.717308
...	...	...
0.9	0.946667	0.058096
1.0	0.946667	0.056398
1.0	1.000000	0.000270

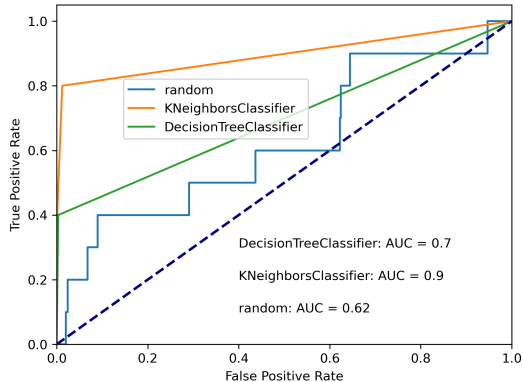
Precision	Recall	Threshold
0.010216	0.9	0.037235
0.010227	0.9	0.037284
0.010239	0.9	0.038246
...	...	...
0.010870	0.8	0.206335
0.010884	0.8	0.206341
...	...	...
0.010870	0.7	0.288363
...	...	...
1.000000	0.0	0.999557

ROC is designed for class-balanced data

When we have 50% positive and 50% negative labels, a line that goes along the diagonal indicates random guess (P=900,N=900).

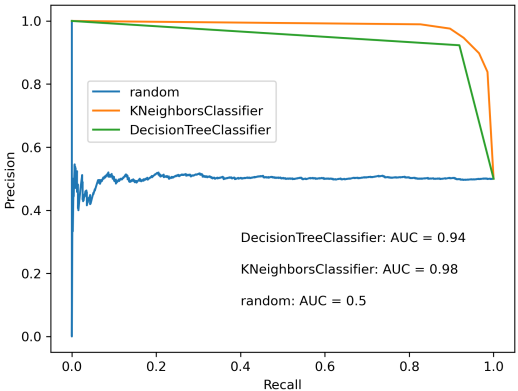


However, suppose we have 1% positive and 99% negative labels, random prediction will no longer follow the diagonal line (P=10,N=900).

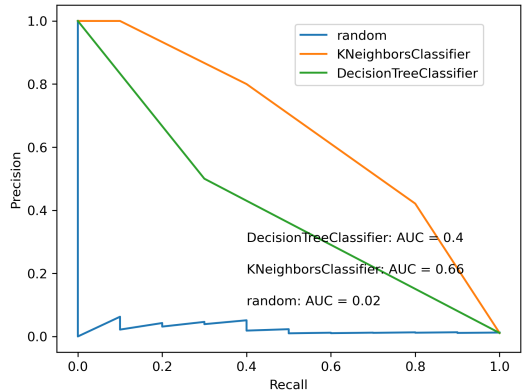


Precision-recall curve is a better choice for imbalanced data

When we have 50% positive and 50% negative labels, a line that goes along the diagonal indicates random guess (P=900,N=900).

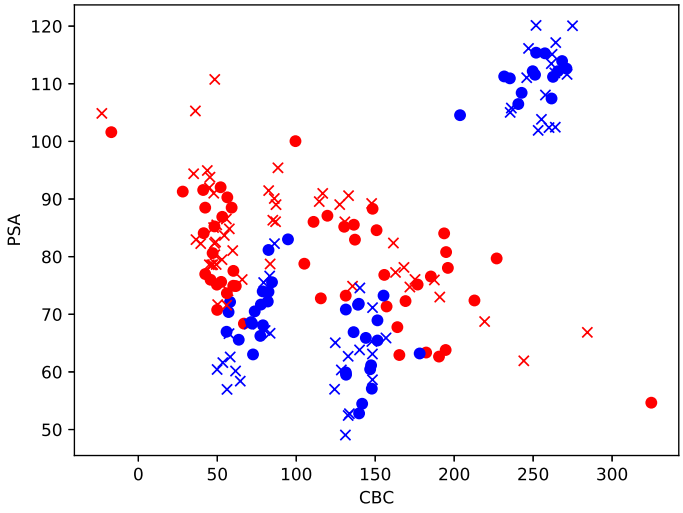


However, suppose we have 1% positive and 99% negative labels, random prediction will no longer follow the diagonal line (P=10,N=900). Here random is close to 0.



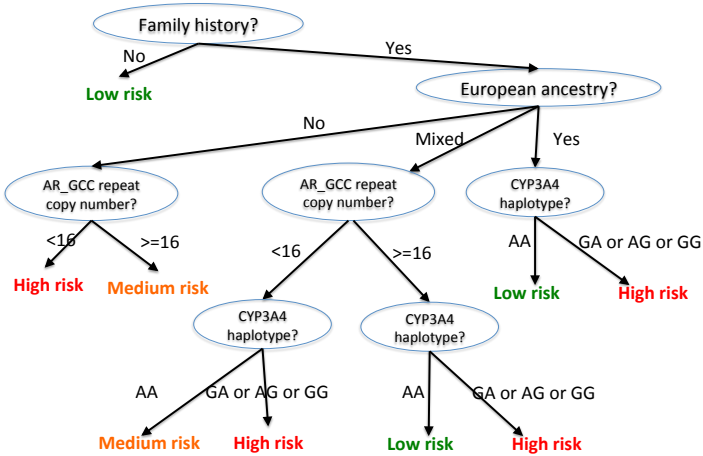
Outline

Non-linearly separable data



Decision tree (DT)

Decision tree is a non-linear classifier. They were often built empirically as a rule-based approach (see the tree below for a toy DT diagnosing non-small cell lung cancer).



Three common losses at each tree node of DT

Misclassification error:

$$e_c = \frac{1}{N_c} \sum_{n=1}^{N} \mathbb{I}(y^{(n)} \neq c) \qquad \text{(error for each class)} \tag{4}$$

$$e = \sum_{c=1}^{C} e_c \qquad \text{(total error)} \tag{5}$$

Entropy loss: $h = -\sum_{c=1}^{C} e_c \log e_c$

Gini loss:

$$\pi_c = \frac{1}{N_c} \sum_{n=1}^{N} \mathbb{I}(y^{(n)} = c) \qquad \text{(class fraction)} \tag{6}$$

$$gini = \sum_c \pi_c (1 - \pi_c) = \sum_c \pi_c - \sum_c \pi_c^2 = 1 - \sum_c \pi_c^2 \tag{7}$$

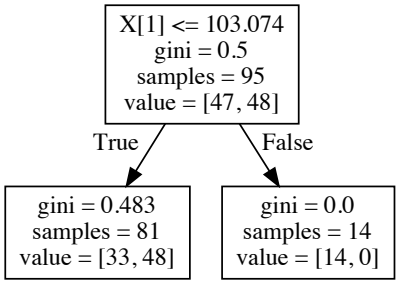
All 3 losses quantify the "impurity" of the tree node. Training a DT is involves minimizing one of the three losses. Gini index is used as default under scikit-learn.

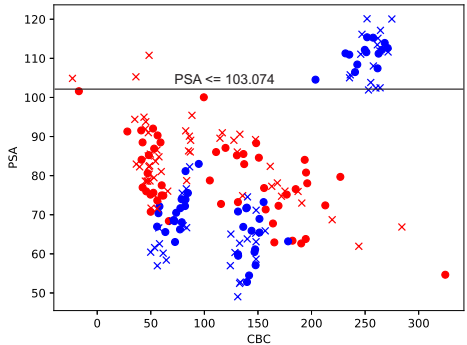
Decision tree using Scikit-Learn

Note: Requires installing graphviz: `pip install graphviz` `tree_slides.py`

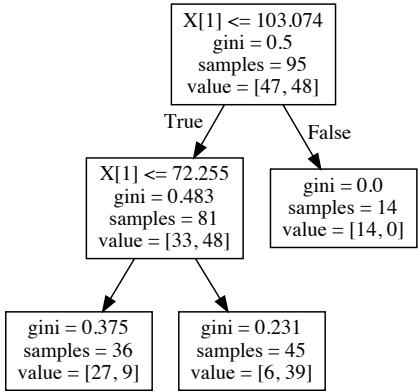
```
1  from sklearn import model_selection,tree
2  import graphviz
3  for depth in range(1,6):
4      clf = tree.DecisionTreeClassifier(max_depth=depth)
5      clf.fit(X_train, y_train)
6      p_train = clf.predict(X_train)
7      p_test = clf.predict(X_test)
8      #plot tree
9      dot_data = tree.export_graphviz(clf, out_file=None)
10     graph = graphviz.Source(dot_data)
11     graph.render("prostate_tree_depth_"+str(depth))
```

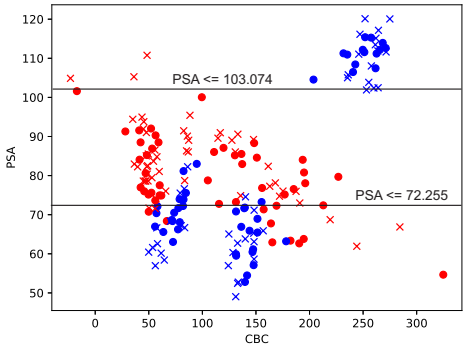
Decision tree at depth = 1



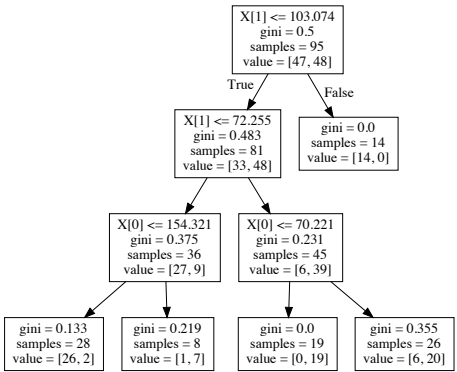


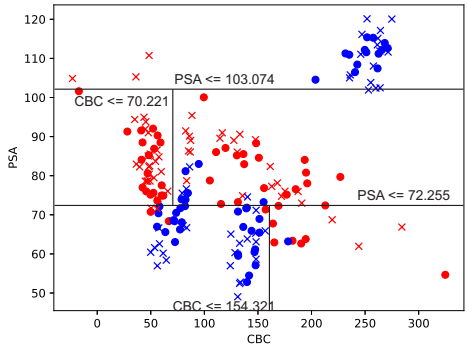
Decision tree at depth = 2



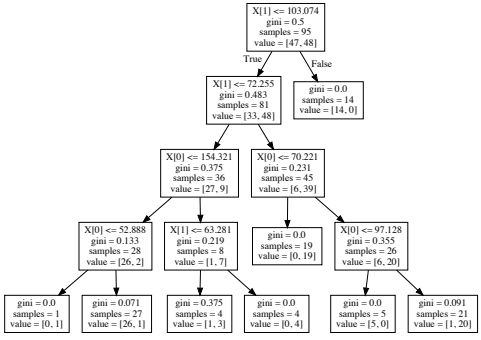


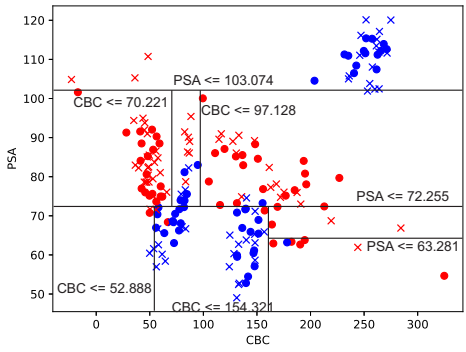
Decision tree at depth = 3



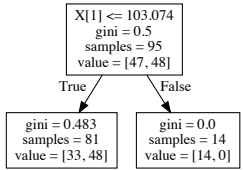


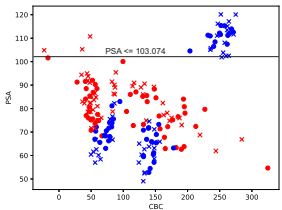
Decision tree at depth = 4





How does DT produce probabilities?





- For a testing example, the predicted class probability is the fraction of *training samples* of the same class in a leaf that contains the test example.
- The left leaf has $33/81 = 0.407$ normal and $48/81 = 0.592$ cancer samples
- The right leaf has $14/14 = 1$ normal and 0 cancer samples
- Suppose the threshold is 0.5 above which we predict the sample as cancer otherwise normal.
- Then, if a patient sample has PSA = 150, what will be the predicted label?
- If a patient sample has PSA = 100, what will be the predicted label? What will be the predicted label at threshold 0.6?

DT greedily chooses the best feature and threshold at each tree node

Algorithm 1 greedy_test(node)

```
min_loss_all = ∞
for each f doeature d ∈ {1, ..., D}
    min_loss_d = ∞
    best_feature_thres = 0
    for each t dohreshold t ∈ {1, ..., T} for feature d
        left_logical = node.X[:, d] ≤ t; loss_left = loss(node.y[left_logical])
        right_logical = node.X[:, d] > t; loss_right = loss(node.y[right_logical])
        loss_d = loss_left + loss_right
        if loss_d < min_loss_d then
            min_loss_d = loss_d
            best_threshold_d = threshold
        end if
    end for
    if min_loss_d < min_loss_all then
        min_loss_all = min_loss_d
        best_feature_index = d
        best_feature_thres = best_threshold_d
    end if
end for
return min_loss_all, best_feature_index, best_feature_thres
```

DT recursively finds the best feature and threshold at every node

Algorithm 2 recursive_fit(node, max_depth, min_leaf_instance)

```
if node.depth < max_depth and len(node.data_indices) > min_leaf_instance then
    split_loss, split_feat, split_value = greedy_test(node)
    test_rule = node.X[node.data_indices, split_feat] <= split_value
    node.split_feature = split_feat
    node.split_value = split_value
    left_node = Node(node.data_indices[test_rule], node)
    right_node = Node(node.data_indices[!test_rule], node)
    recursive_fit(left_node)
    recursive_fit(right_node)
     node.left = left_node
     node.right = right_node
end if
```

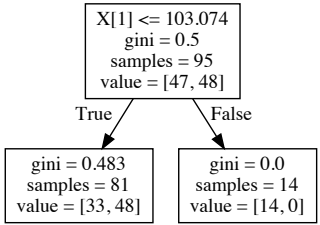
Algorithm 3 fit(tree, max_depth, min_leaf_instance)

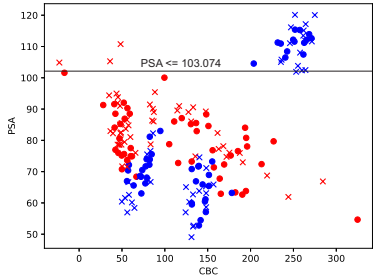
```
recursive_fit(tree.root, max_depth, min_leaf_instance)
```

See Colab for Python implementation of the above algorithm.

Outline

Decision tree (max_depth = 1)





Training data:

	PN	PP
AN	14	33
AP	0	48

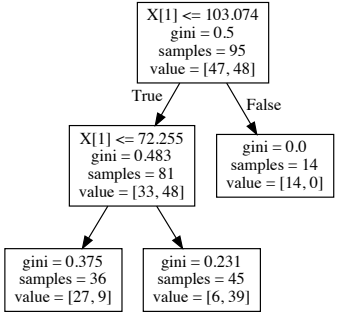
TPR = 48/(48+0) = 1.0
FPR = 33/(33+14) = 0.7

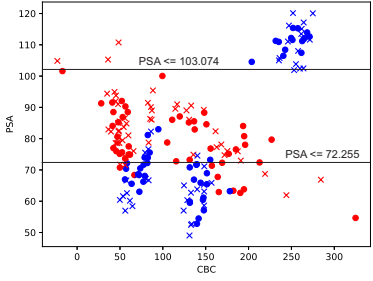
Test data:

	PN	PP
AN	13	30
AP	3	49

TPR = 49/(49+3) = 0.94
FPR = 30/(30+13) = 0.7

Decision tree (max_depth = 2)





Training data:

	PN	PP
AN	41	6
AP	9	39

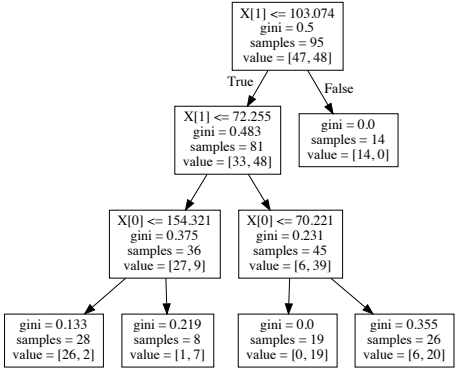
TPR = 39/(39+9) = 0.81
FPR = 6/(6+41) = 0.13

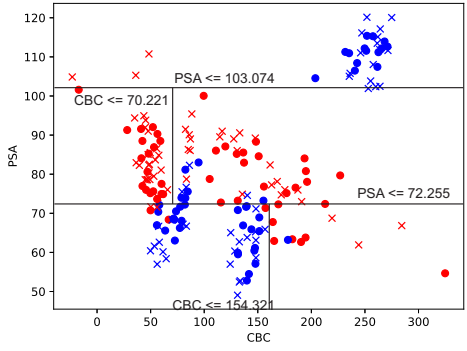
Test data:

	PN	PP
AN	36	7
AP	8	44

TPR = 44/(44+8) = 0.85
FPR = 7/(7+36) = 0.16

Decision tree (max_depth = 3)





Training data:

	PN	PP
AN	40	7
AP	2	46

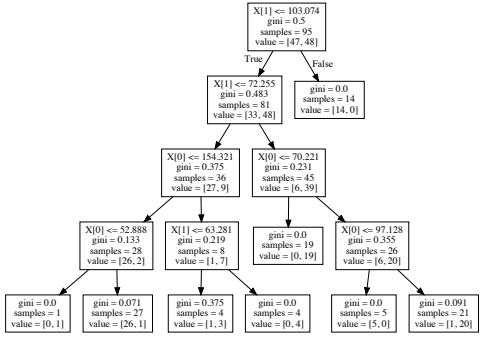
TPR $= 46/(46+2) = 0.96$
FPR $= 7/(7+40) = 0.15$

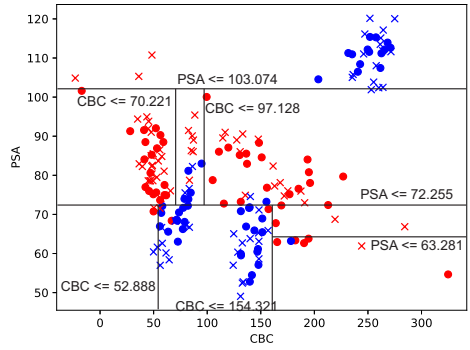
Test data:

	PN	PP
AN	35	8
AP	5	47

TPR $= 47/(47+5) = 0.9$
FPR $= 8/(8+35) = 0.19$

Decision tree (max_depth = 4) - overfitting occurs





Training data:

	PN	PP
AN	45	2
AP	1	47

TPR $= 47/(47+1) = 0.98$
FPR $= 2/(2+45) = 0.04$

Test data:

	PN	PP
AN	37	6
AP	11	41

TPR $= 41/(41+11) = 0.79$
FPR $= 6/(6+37) = 0.14$

Decision tree (max_depth = 5 & 6) - more overfitting

Tree depth = 5

Training data:

	PN	PP
AN	46	1
AP	1	47

$TPR = 47/(47+1) = 0.98$

$FPR = 1/(1+46) = 0.02$

Test data:

	PN	PP
AN	37	6
AP	11	41

$TPR = 41/(41+11) = 0.79$

$FPR = 6/(6+37) = 0.14$

Tree depth = 6

Training data:

	PN	PP
AN	47	0
AP	0	48

$TPR = 48/(48+0) = 1.0$

$FPR = 0/(0+47) = 0.0$

Test data:

	PN	PP
AN	37	6
AP	11	41

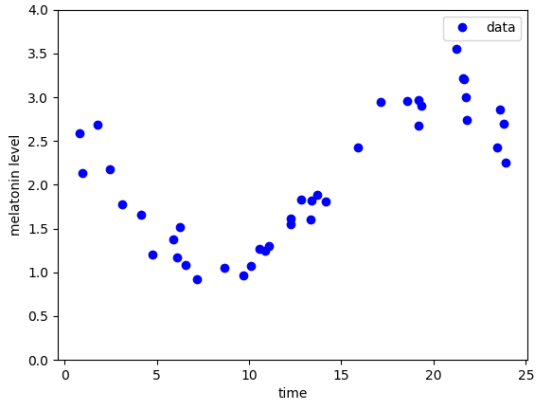
$TPR = 41/(41+11) = 0.79$

$FPR = 6/(6+37) = 0.14$

Outline

A regression problem

Background: Melatonin (sleep hormone) levels vary over time in a cyclical manner.
Data: We have measured the patient's melatonin levels at different times.
Goal: Learn to predict a patient's melatonin level as a function of time, e.g. to choose when to deliver a drug



Regression problem

- Problem: Let the melatonin level be y and time be x
- Goal: Learn a function $f(x)$ to predict y values from x values
- Objective function: sum of square errors

$$E = \sum_{i \in train} (f(x_i) - y_i)^2$$

- Algorithm: We will treat $f(x)$ as a *polynomial*:
- Let's start with a polynomial of degree 1:

$$f(x) = ax + b$$

- The goal of learning is to choose the value of coefficients a and b based on training data.
- We want to choose a and b so as to best fit the training data that *minimizes* the sum of square error

$$a, b \leftarrow \arg\min_{a,b} E$$

Regression with scikit-learn

To learn a regression using scikit-learn:

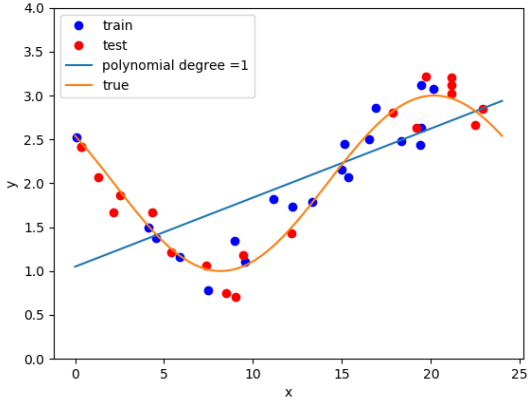
```
# transform data into matrices for regression
reg_X_train = X_train[:,np.newaxis]
reg_X_test = X_test[:,np.newaxis]

# plot regression analysis
for degree in [1,2, 5,10,15,20]:
    # Create a polynomial regression model
    model = make_pipeline(PolynomialFeatures(degree), Ridge(0))

    # Fit the model to the training data
    model.fit(reg_X_train, y_train)

    # Apply the model to make predictions on the training data
    pred_train =  model.predict(reg_X_train)

    # Apply the model to make predictions on the test data
    pred_test =  model.predict(reg_X_test)

    # Calculate mean squared errors
    train_err = mean_squared_error(y_train,pred_train)
    test_err = mean_squared_error(y_test,pred_test)
```

1-degree polynomial regression

For our data, the best choice is $a = 1.4$, $b = 0.9$ i.e.

$$y = 1.4x + 0.9$$

Just by eyeballing, we can tell that the fit is not good.



Mean Squared error and Underfitting

Problem: Just by looking at the plot we can tell that the fit to the training data is very bad:
The fitted line is far from the observed values at most training examples.
Measuring prediction errors:
Mean-squared-error = Sum of the squares of the difference between the predicted and observed values divided by the total number of the training examples:

$$MSE(train) = \frac{\sum_{i \in train} (f(x_i) - y_i)^2}{N_{train}}$$

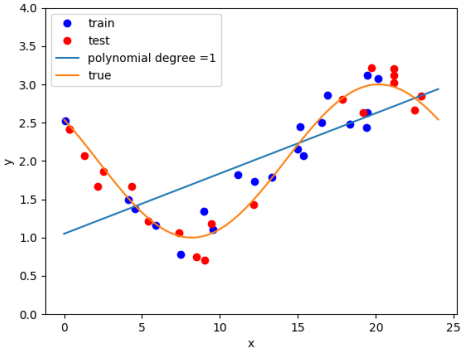
Here: MSE(train) = 0.31 and MSE(test) = 0.50

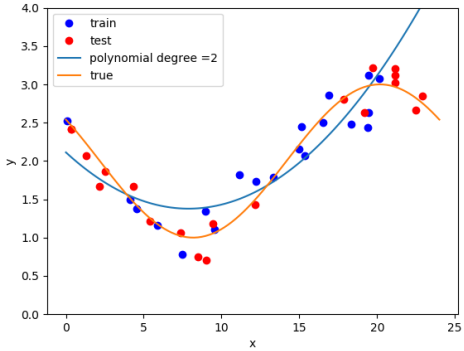
When the training error is too large, we call this *underfitting*: The predictor cannot fit the training data well because it is too limited in the type of functions it can represent.

Some improvement with 2-degree polynomial

We can improve the fit to the training data by considering a polynomial of degree 2:

$$f(x) = ax^2 + bx + c$$

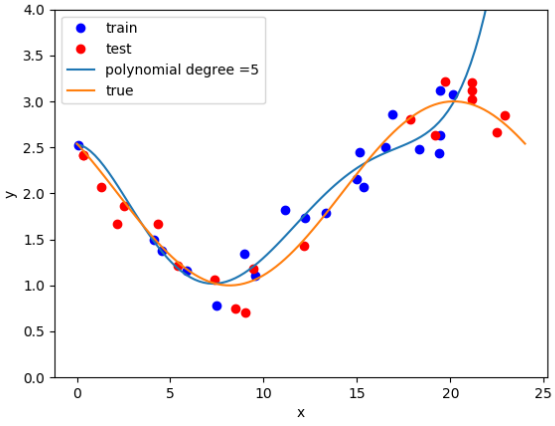




- 1-degree: MSE(train) = 0.31 and MSE(test) = 0.50
- 2-degree: MSE(train) = **0.099**, MSE(test) =**0.23** (somehow testing error is slightly higher than 1-degree)

Higher-degree polynomial

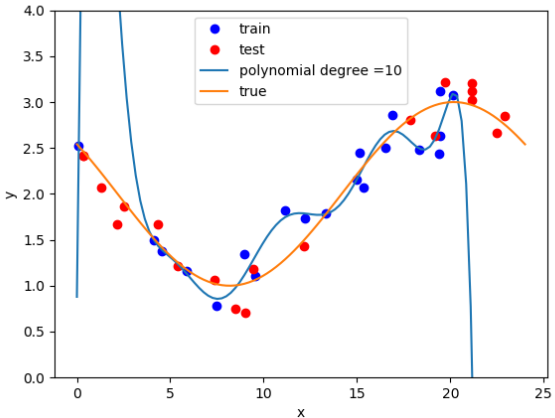
We can further improve the fit to the training data by considering higher degree polynomial, e.g. degree $= 5$ $f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$



- 1-degree: MSE(train) $= 0.31$ and MSE(test) $= 0.50$
- 2-degree: MSE(train) $=$ 0.099, MSE(test) $=0.23$
- 5-degree: MSE(train) $=$ **0.022**, MSE(test) $=$**0.057**

And . . . even higher-degree polynomial

Let's see if we keep going to higher degrees: degree = 10



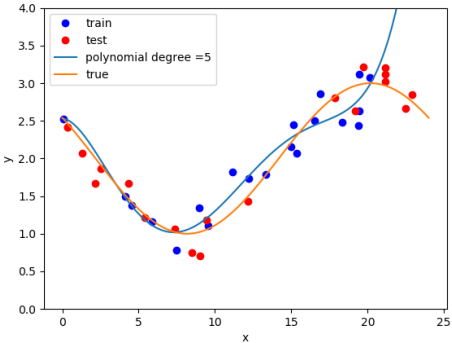
- 1-degree: MSE(train) = 0.442 and MSE(test) = 0.545
- 2-degree: MSE(train) = 0.42, MSE(test) =0.598
- 5-degree: MSE(train) = 0.028, MSE(test) =**0.066**
- 10-degree: MSE(train) = **0.01**, MSE(test) =0.40

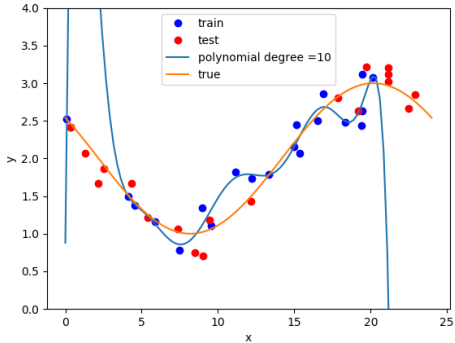
Overfitting

If the number of parameters to learn is large (e.g. for a polynomial of degree 10, there are 11 parameters), the predictor is able to fit the training data very well: MSE(train) is very small.

But the corresponding testing error MSE(test) is very large!

This is *bad*, because our goal is for our predictor to do well on the test data (i.e. data it hasn't seen during its training).





Overfitting

If the number of parameters to learn is large (e.g. for a polynomial of degree 10, there are 11 parameters), the predictor is able to fit the training data very well: MSE(train) is very small.
But the corresponding testing error MSE(test) is very large!
This is *bad*, because our goal
is for our predictor to do well on the test data (i.e. data it hasn't seen during its training).

This is called *overfitting*:Predictor is able to fit the training data very well, but fits testing data very poorly:

$$MSE(train) << MSE(test).$$

Overfitting happens when the predictor has too much flexibility in choose the values of too many parameters.
To limit overfitting, we have to limit the number of parameters the predictor has to estimate (or use other means such as regularization).

Summary

- Precision-recall is an alternative metric to ROC and it is better suited to measure performance on imbalanced data and circumstance where precision is important.
- Decision tree is a greedy and recursive algorithm that find the best feature and the best threshold at each tree node to minimize the loss (e.g., Gini index) within the node.
- Overfitting occurs when the model gets too complicated and fit extremely well on the training data but generalize poorly to the test data.

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.
If you rerun the document (without altering it) this surplus page will go away, becau LaTeX now knows how many pages to expect for this document.