Fondamenti di IA

02 - Model Selection and Theory of Learning

SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

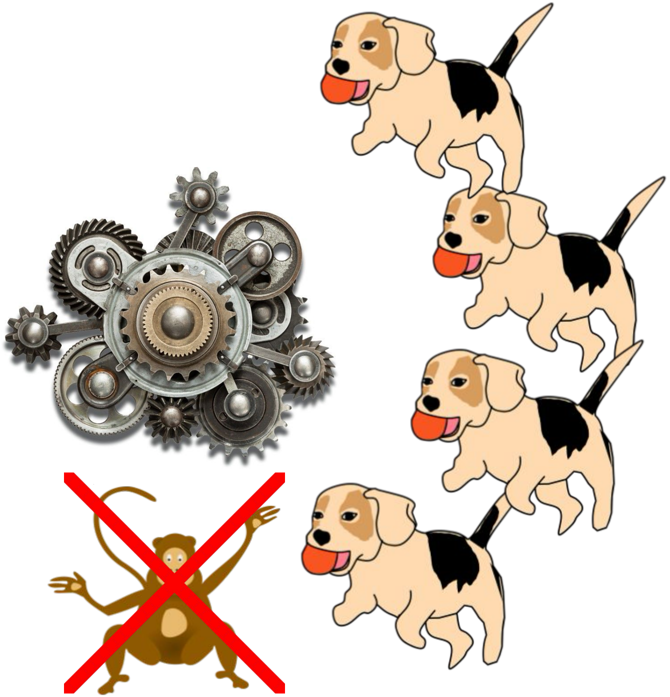
Model Selection

The Goal

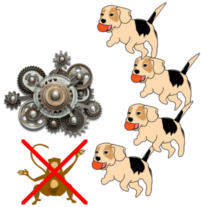
- Our goal in machine learning is to select a hypothesis that will optimally fit future examples.
  - To make that precise we need to define “future example” and “optimal fit.”

Future Examples…

- … will be like the past.



Independent and Identically Distributed – i.i.d.



- We assume that each example E_j has the same prior probability distribution:

$$P(E_j) = P(E_{j+1}) = P(E_{j+2}) = \dots$$ Identically Dist.

and is independent of the previous examples:

$$P(E_j) = P(E_j | E_{j-1}, E_{j-2}, \dots)$$ Independence

This is the most important and frequently made assumption in Machine Learning. From now on, we will **always assume i.i.d.**

SAPIENZA
UNIVERSITÀ DI ROMA

Optimal Fit

- the optimal fit is the hypothesis that minimizes the **error rate**:
  - the proportion of times that h(x) ≠ y for an (x, y) example.
- Simple, huh?
  - Where do you measure that?
  - Is it the only measure of error?
  - Can we measure what **is going to be** the error rate in the future?

Test sets

- We can estimate the error rate of a hypothesis by giving it a **test**:
  - measure its performance on a test set of examples. It would be cheating for a hypothesis (or a student) to peek at the test answers before taking the test.
- The simplest way to ensure this doesn’t happen is to split the examples you have into two sets:
  - a training set to create the hypothesis, and a test set to evaluate it.

Multiple hypothesis

- If we are only going to create one hypothesis, then the approach based on splitting training and test set is sufficient.
- Often we will end up creating multiple hypotheses:
  - we might want to compare two completely different machine learning models, or we might want to adjust the various “knobs” within one model.
  - For example, we could try different thresholds for χ2 pruning of decision trees, or different degrees for polynomials.

Hyperparameter Tuning on Test Datasets

- We call these “knobs” **hyperparameters** — parameters of the model class, not of the individual model.
- Suppose a researcher generates a hypotheses for one setting of the $\chi2$ pruning hyperparameter, measures the error rates on the test set, and then tries different hyperparameters.
    - No individual hypothesis has peeked at the test set data, but the overall process did, through the researcher.

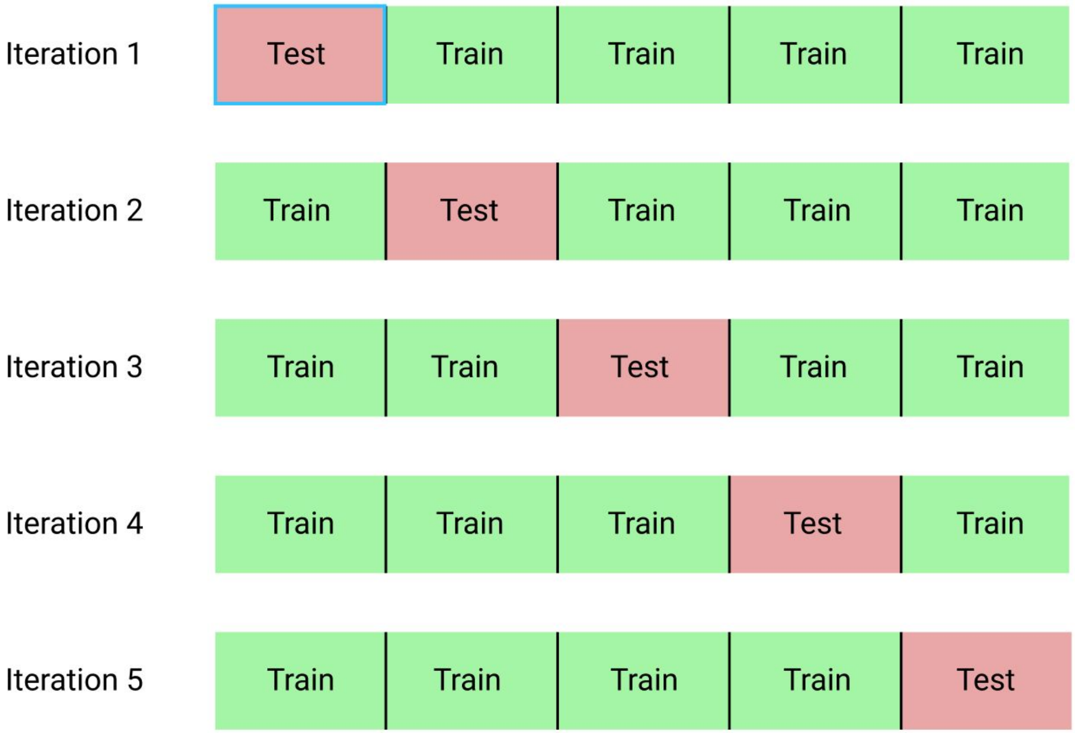
Train, validate, and test

- *No individual hypothesis has peeked at the test set data, but the overall process did, through the researcher.*
- The way to avoid this is to **really hold out the test set** — lock it away until you are completely done with training, experimenting, hyperparameter-tuning, re-training, etc.
- That means you need three data sets:
  - A training set to train candidate models.
  - A validation set, also known as a *development set* or *dev set*, to evaluate the candidate models and choose the best one.
  - A test set to do a final *unbiased evaluation* of the best model.

K-Fold Cross-Validation

- What if we don’t have enough data to make all three of these data sets?
  - We can squeeze more out of the data using a technique called **k-fold cross-validation**.
- The idea is that each example serves double duty — as training data and validation data — but not at the same time.
- First we split the data into ***k*** equal subsets.
- We then perform ***k*** rounds of learning;
  - on each round ***1/k*** of the data are held out as a validation set and the remaining examples are used as the training set.
  - The average test set score of the ***k*** rounds should then be a better estimate than a single score.
    - Popular values for ***k*** are 5 and 10
      - enough to give an estimate that is statistically likely to be accurate, at a cost of 5 to 10 times longer computation time.
- Even with cross-validation, we still need a separate test set.

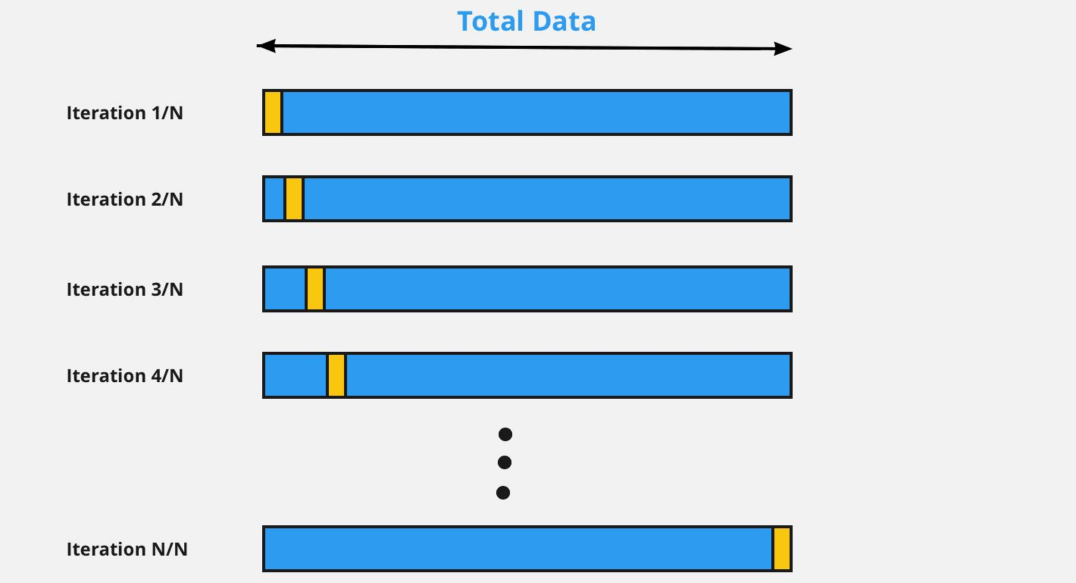
K-Fold Cross-Validation



Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

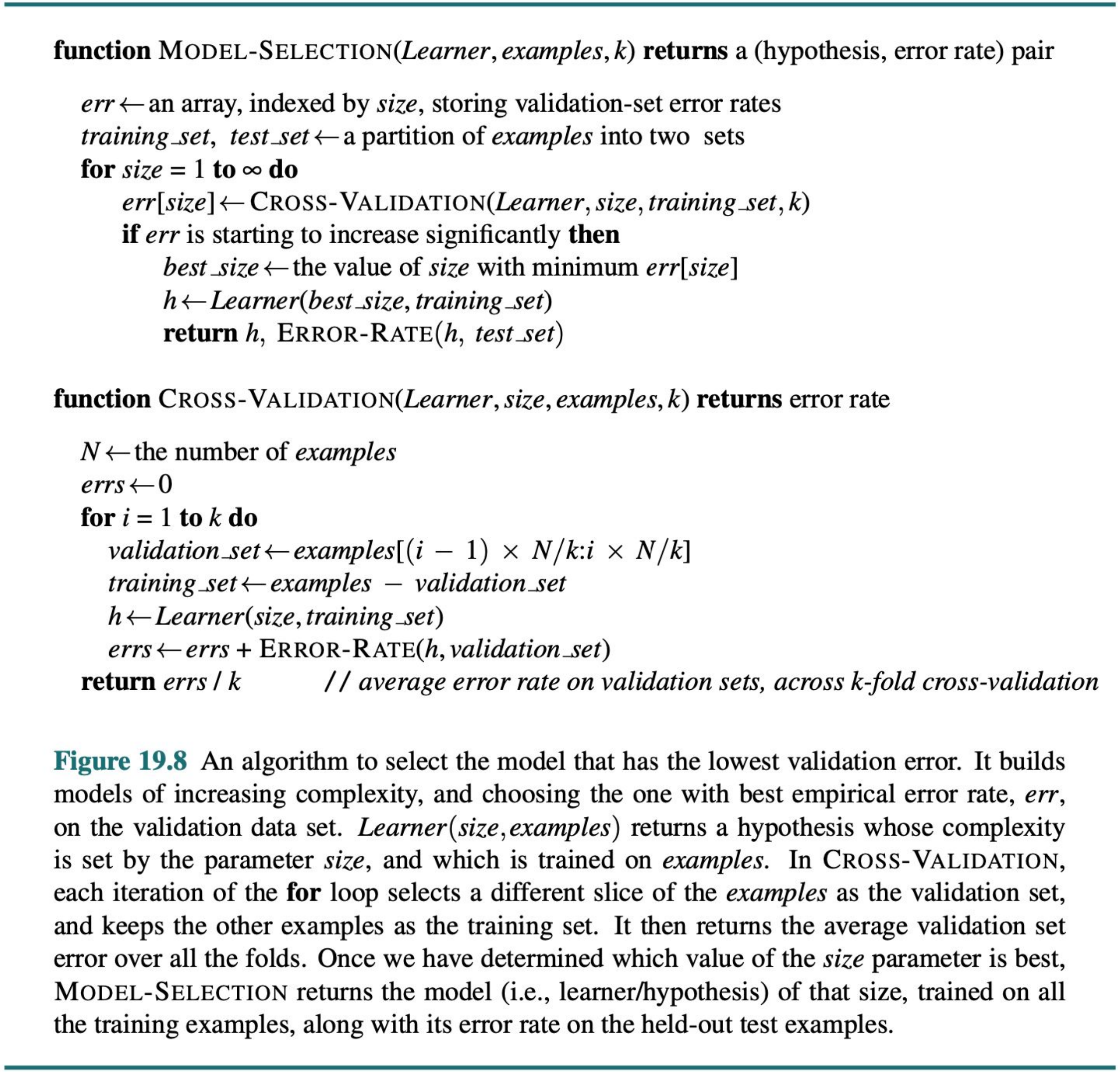
Leave One Out Cross Validation

- The extreme is of k-fold cross-validation is k = n
  - leave-one-out cross-validation or LOOCV.



Model Selection vs. Optimization

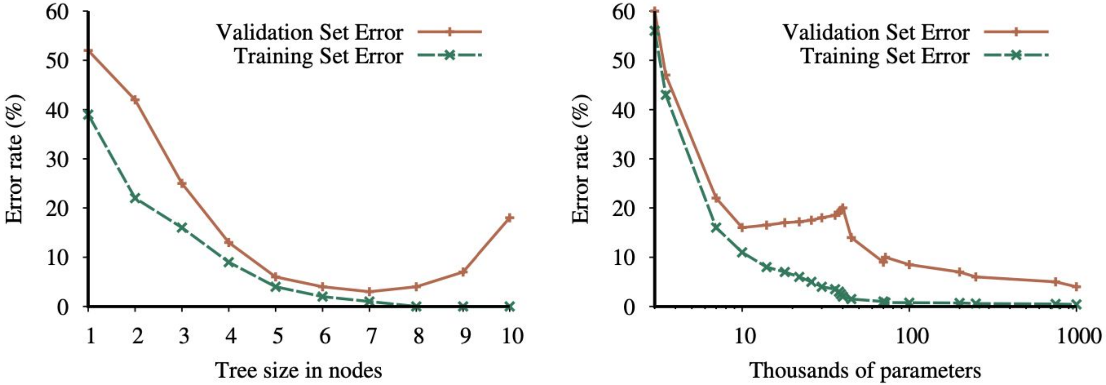
- We can think of the task of finding a good hypothesis as two subtasks:
  - model selection → chooses a good hypothesis space, and
  - optimization (also called training) → finds the best hypothesis within that space.
- Part of model selection is qualitative and subjective:
  - we might select polynomials rather than decision trees based on something that we know about the problem.
- Part is quantitative and empirical: within the class of polynomials, we might select Degree = 2, because that value performs best on the validation data set.



```
function MODEL-SELECTION(Learner, examples, k) returns a (hypothesis, error rate) pair

  err ← an array, indexed by size, storing validation-set error rates
  training_set, test_set ← a partition of examples into two sets
  for size = 1 to ∞ do
      err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)
      if err is starting to increase significantly then
          best_size ← the value of size with minimum err[size]
          h ← Learner(best_size, training_set)
          return h, ERROR-RATE(h, test_set)

function CROSS-VALIDATION(Learner, size, examples, k) returns error rate

  N ← the number of examples
  errs ← 0
  for i = 1 to k do
     validation_set ← examples[$(i - 1) \times N/k : i \times N/k$]
     training_set ← examples − validation_set
     h ← Learner(size, training_set)
     errs ← errs + ERROR-RATE(h, validation_set)
  return errs / k          // average error rate on validation sets, across k-fold cross-validation
```

Figure 19.8 An algorithm to select the model that has the lowest validation error. It builds models of increasing complexity, and choosing the one with best empirical error rate, *err*, on the validation data set. *Learner*(*size*, *examples*) returns a hypothesis whose complexity is set by the parameter *size*, and which is trained on *examples*. In CROSS-VALIDATION, each iteration of the **for** loop selects a different slice of the *examples* as the validation set, and keeps the other examples as the training set. It then returns the average validation set error over all the folds. Once we have determined which value of the *size* parameter is best, MODEL-SELECTION returns the model (i.e., learner/hypothesis) of that size, trained on all the training examples, along with its error rate on the held-out test examples.

Error rates



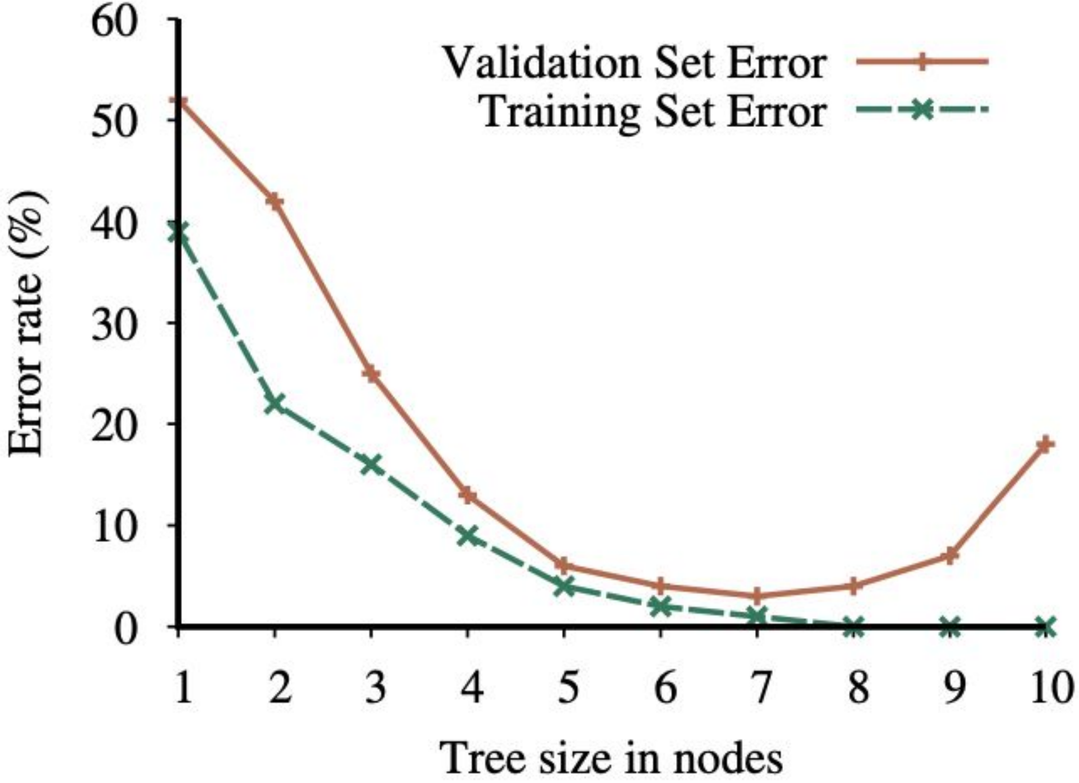
(a) (b)

Figure 19.9 Error rates on training data (lower, green line) and validation data (upper, orange line) for models of different complexity on two different problems. MODEL-SELECTION picks the hyperparameter value with the lowest validation-set error. In (a) the model class is decision trees and the hyperparameter is the number of nodes. The data is from a version of the restaurant problem. The optimal size is 7. In (b) the model class is convolutional neural networks (see Section 22.3) and the hyperparameter is the number of regular parameters in the network. The data is the MNIST data set of images of digits; the task is to identify each digit. The optimal number of parameters is 1,000,000 (note the log scale).

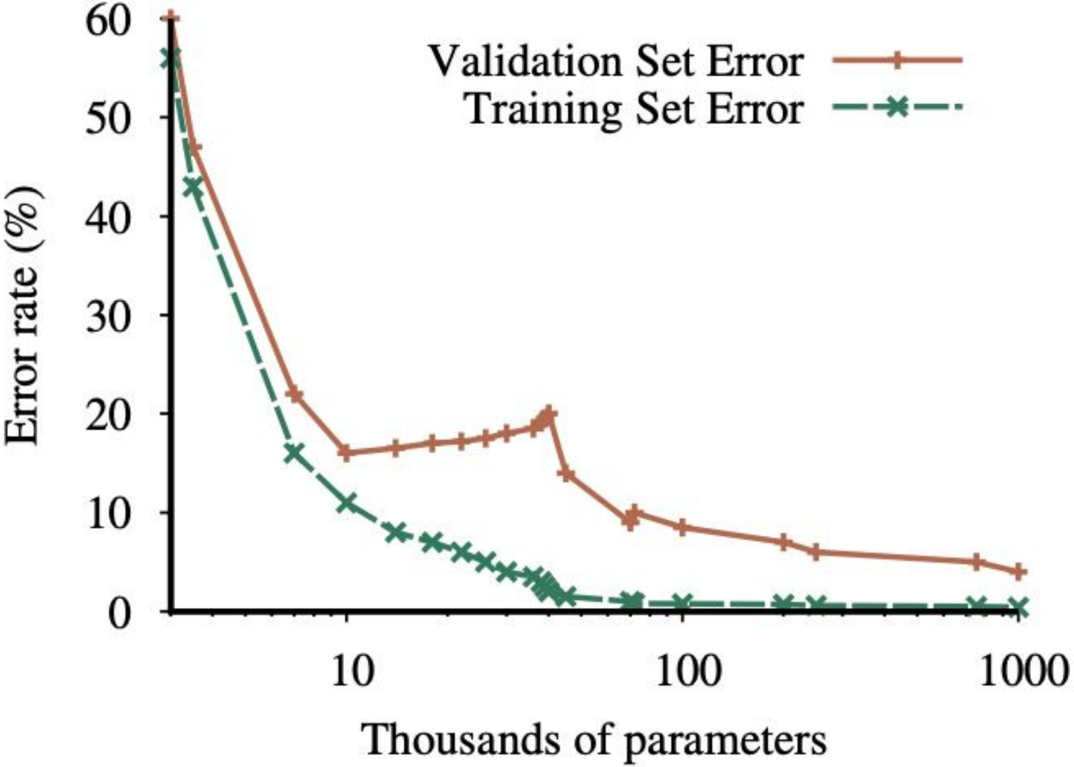
Complexity of a model

- Complexity is measured differently depending on the family of hypothesis.
- For example:
  - the number of decision tree nodes
  - the number of parameters in a polynomial (same as degree, why?)
- For many model classes, *the training set error reaches zero as the complexity increases*.

U-Shaped Behavior



Double dip



Validation Set Error
Training Set Error
Error rate (%)
60
50
40
30
20
10
0
10
100
1000
Thousands of parameters

Interpolation vs. Extrapolation

- A Decision tree with n leaves can represent perfectly a sample of n observations.
- A polynomial of degree n can represent perfectly a sample of n + 1 observations.
- We say that a model that exactly fits all the training data has interpolated the data. Model classes typically start to overfit as the capacity approaches the point of interpolation.
- Some model classes never recover from this overfitting, as with the decision trees it.
- For other model classes, adding capacity means that there are more candidate functions, and some of them are naturally well-suited to the patterns of data that are in the true function.
  - The higher the capacity, the more of these suitable representations there are, and the more likely that the optimization mechanism will be able to land on one.

From Error Rate to Loss

- So far, we have been trying to minimize error rate. This is clearly better than maximizing error rate, but it is not the full story.
  - Consider the problem of classifying email messages as spam or non-spam.
    - It is worse to classify non-spam as spam but we can tolerate the other way around
    - So a classifier with a 1% error rate, where almost all the errors were classifying spam as non- spam, would be better than a classifier with only a 0.5% error rate, if most of those errors were classifying non-spam as spam.

Loss Function

- The goal of ML is to **minimize a loss function** (maximize the utility).
- Let x be a sample, y be the label associated with x (y = f(x)) and ŷ the label that our model h produces → We represent by L(x, y, ŷ) the amount of utility lost by predicting h(x) = ŷ when the correct answer is f(x) = y
  - For our purposes we will not make the loss depend on **x → L(y, ŷ)**
- This allows us to compute things such as:
  - L(spam, nospam) = 1, L(nospam, spam) = 10
  - Note that L(y,y) is always zero

Some Examples of Loss Functions

- If f(x) is 137.035999, we would be fairly happy with h(x) = 137.036, but just how happy should we be?
  - In general, small errors are better than large ones; two functions that implement that idea are the absolute value of the difference (called the L1 loss), and the square of the difference (called the L2 loss).
- For discrete-valued outputs, if we are content with the idea of minimizing error rate, we can use the $L_{0/1}$ loss function, which has a loss of 1 for an incorrect answer.
- Absolute-value loss: $L_1(y,\hat{y})=|y-\hat{y}|$
- Squared-error loss: $L_2(y, \hat{y}) = (y - \hat{y})^2$
- 0/1 loss: $L_{0/1}(y,\hat{y})$ = 0 if $y = \hat{y}$, else 1

Generalization Loss

- Theoretically, the learning agent maximizes its expected utility by choosing the hypothesis that minimizes expected loss over all input–output pairs it will see.
- To compute this expectation we need to define a prior probability distribution P(X,Y) over examples.
- Let $\mathcal{E}$ be the set of all possible input–output examples. Then the expected **generalization loss** for a hypothesis h (with respect to loss function L) is

$$GenLoss_L(h) = \sum_{(x,y)\in\mathcal{E}} L(y,h(x))\,P(x,y)$$

- and the best hypothesis, h*, is the one with the minimum expected generalization loss:

$$h^* = \operatorname*{argmin}_{h\in\mathcal{H}} GenLoss_L(h)$$

Empirical Loss

- Because P(x,y) is not known in most cases, we can only estimate generalization loss with empirical loss on a set of examples E of size N

$$EmpLoss_{L,E}(h) = \sum_{(x,y)\in E} L(y, h(x)) \frac{1}{N}$$

- The estimated best hypothesis ĥ* is then the one with minimum empirical loss

$$\hat{h}^* = \operatorname*{argmin}_{h\in\mathcal{H}} EmpLoss_{L,E}(h)$$

- There are four reasons why ĥ* may differ from the true function, f: **unrealizability**, variance, noise, and **computational complexity**.

Realizable

- A learning problem is realizable iff the hypothesis space $\mathcal{H}$ actually contains the true function f that generated the data.
  - E.g., if $\mathcal{H}$ is the set of linear functions, and the true function f is a quadratic function, then no amount of data will recover the true f.

Variance

- Variance means that a learning algorithm will in general return different hypotheses for different sets of examples.
- If the problem is realizable, then variance decreases towards zero as the number of training examples increases.

Noise

- f may be nondeterministic or noisy (it may return different values of f(x) for the same x).
- Noise cannot be predicted (it can only be characterized).

Computational Complexity

- When $\mathcal{H}$ is a complicated function in a large hypothesis space, it can be computationally intractable to systematically search all possibilities.

Small-scale learning

- Traditional methods in statistics and the early years of machine learning concentrated on small-scale learning
  - the number of training examples ranged from dozens to the low thousands.
- The generalization loss mostly comes from
  - the approximation error of not having the true f in the hypothesis space
  - the estimation error of not having enough training examples to limit variance.

Large-scale learning

- In recent years there has been more emphasis on large-scale learning, with millions of examples.
- The generalization loss may be dominated by limits of computation:
  - there are enough data and a rich enough model that we could find an h that is very close to the true f
  - but the computation to find it is complex, so we settle for an approximation.

Penalizing Complex Hypothesis

- Model selection is used to select the best hypothesis.
- One alternative way is that of minimize the empirical loss **and** the complexity of the model
  - A complex model will more easily overfit.
- The weighted sum of these two components

$$\begin{aligned} Cost(h) &= EmpLoss(h) + \lambda\, Complexity(h) \\ \hat{h}^* &= \operatorname*{argmin}_{h \in \mathcal{H}} Cost(h)\,. \end{aligned}$$

  is called total cost.
- Here λ is a hyperparameter
  - a positive number that serves as a conversion rate between loss and hypothesis complexity.

Regularization

- Is the process of explicitly penalizing complex hypotheses
    - we’re looking for functions that are more regular.
- Note that we are now making two choices: the loss function (L1 or L2), and the complexity measure, which is called a regularization function.
- The choice of regularization function depends on the hypothesis space.
    - For example, for polynomials, a good regularization function is the sum of the squares of the coefficients
    - keeping the sum small would guide us away from a complex polynomial by forcing some coefficients to zero.

Feature Selection

- Regularization can be used to select the features that really matter to a model.
- χ2 pruning for Decision Trees is also a kind of feature selection
- There are also method that are specifically designed for selecting the best features of a model
  - Highest coefficients
  - Correlation of coefficients
  - …

Hyperparameter Tuning

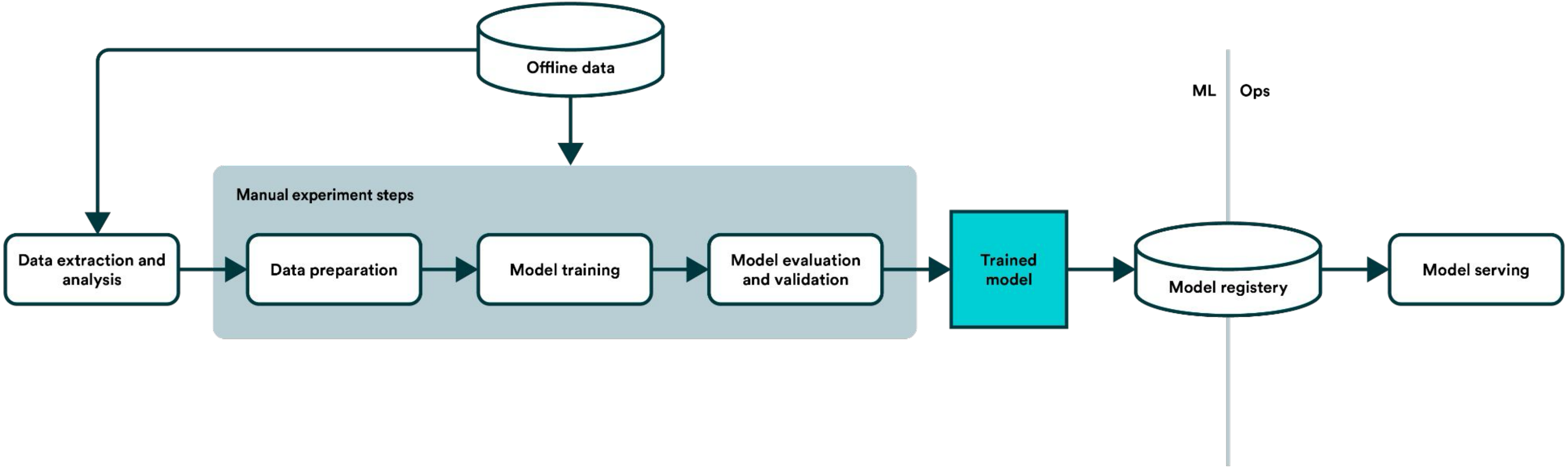
- Hand Tuning
  - Guess some parameter values based on past experience, train a model, measure its performance on the validation data, analyze the results, and use your intuition to suggest new parameter values.
  - Repeat until you have satisfactory performance (or you run out of time, computing budget, or patience).
- Grid Search
  - Try all combinations of values and see which performs best on the validation data.
  - Different combinations can be run in parallel on different machines, so if you have sufficient computing resources, this can be fast enough.
- Random Search
  - If there are too many combinations of possible values, then random search samples uniformly from the set of all possible hyperparameter settings, repeating for as long as you are willing to spend the time and computational resources.

In practice

Regularization

Picture taken from: https://valohai.com/machine-learning-pipeline/

What does an ML Pipeline look like?



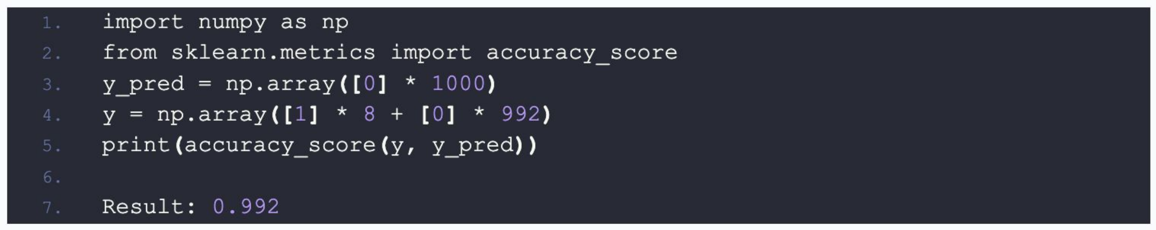
Evaluating an ML Task

- There are several evaluation metrics used in ML and each metric depends on the task at hand:
    - Classification
        - Accuracy
        - Confusion matrix
        - False Positives and False Negatives
        - Precision and Recall
        - F1 Score
        - ROC Curve
        - Precision Recall Curve
        - AUC
    - Regression
        - Mean Absolute Error (MAE)
        - Mean Squared Error (MSE)
        - Root Mean Squared Error on Prediction (RMSE / RMSEP)

Accuracy

- Accuracy gives us an overall picture of how much we can rely on our model’s prediction.
- This metric is blind to the difference between classes and types of errors, hence for imbalanced datasets accuracy, it is generally not enough.

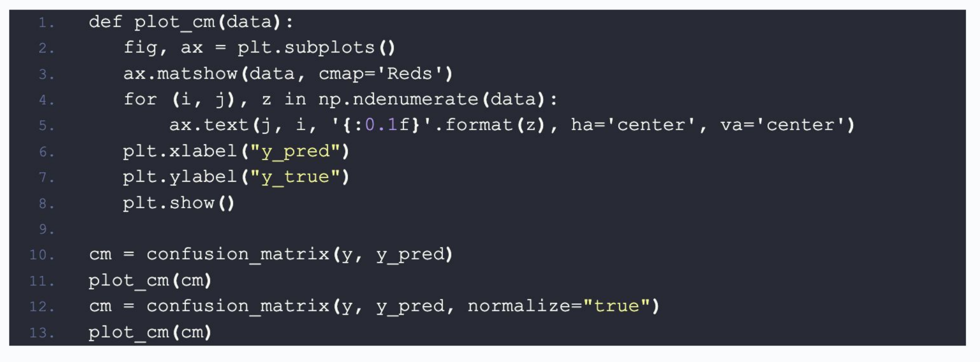
$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$



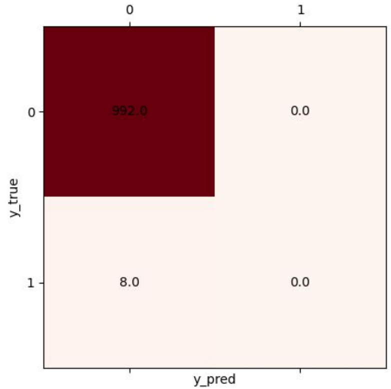
```
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = np.array([0] * 1000)
y = np.array([1] * 8 + [0] * 992)
print(accuracy_score(y, y_pred))

Result: 0.992
```

Confusion Matrix

- One way to ensure we are not blinded by the overall accuracy is to evaluate our model quality on each class independently.
- A popular way to visualize this is by using the confusion matrix.



```python
def plot_cm(data):
    fig, ax = plt.subplots()
    ax.matshow(data, cmap='Reds')
    for (i, j), z in np.ndenumerate(data):
        ax.text(j, i, '{:0.1f}'.format(z), ha='center', va='center')
    plt.xlabel("y_pred")
    plt.ylabel("y_true")
    plt.show()

cm = confusion_matrix(y, y_pred)
plot_cm(cm)
cm = confusion_matrix(y, y_pred, normalize="true")
plot_cm(cm)
```



False Positives and False Negatives

- A false positive is an error in binary classification in which a test result incorrectly indicates the presence of a condition such as a disease when the disease is not present
- A false negative is the opposite error where the test result incorrectly fails to indicate the presence of a condition when it is present.

Precision and Recall

- Precision and Recall are two popular metrics that do contribute to our understanding of the types of errors we have.
- Precision or positive predictive value gives us a measure for how much we can trust a positive prediction of our model.
- Recall, sensitivity or true positive rate (TPR) gives us a measure for how many of the real “true” values we detected.

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

$$TPR = Recall = \frac{TP}{TP + FN}$$

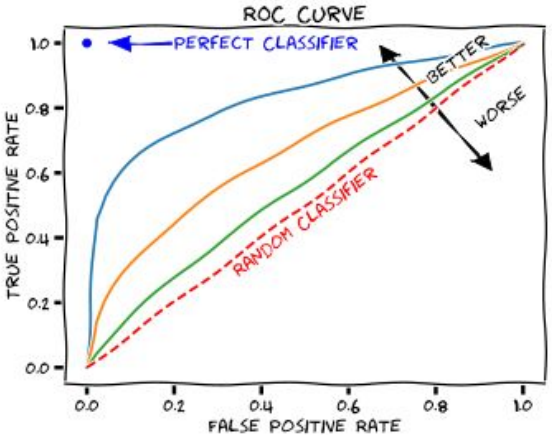
F1 Score

- Usually, there is a tradeoff between getting high precision and high recall, thus a common metric that gives a balanced overall score is the F1 score.
    - F1 is defined as the harmonic mean of the precision and recall.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

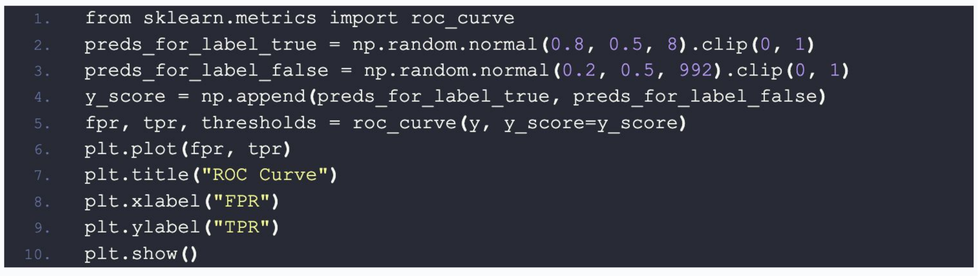
ROC Curve

- In order to visualize the tradeoff between false positives and false negatives for a given model, we can plot the receiver operating characteristic (ROC) curve
  - It plots different possible values of TPR and FPR that are obtained by using different decision thresholds (the threshold for deciding whether a prediction is labeled “true” or “false”) for the predictive model

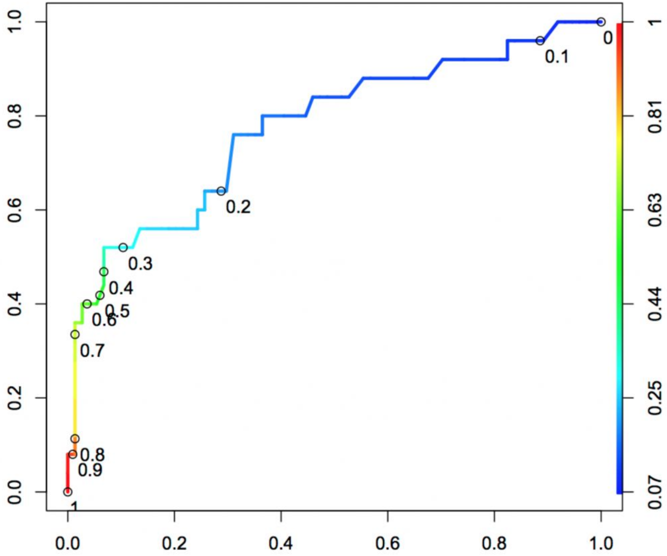


ROC Curve

- Example in scikit learn

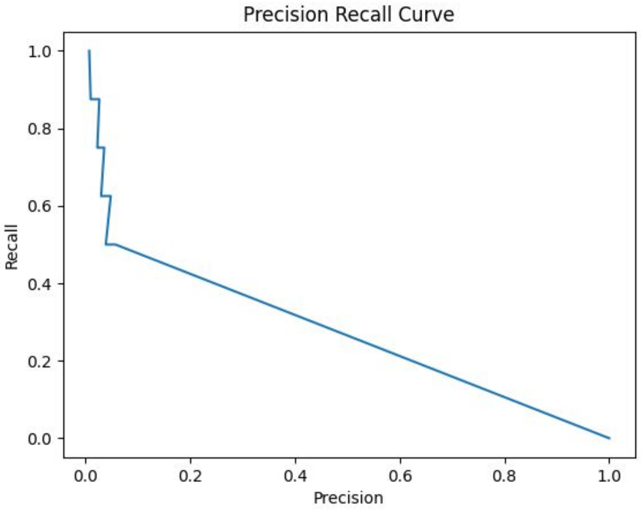


```python
from sklearn.metrics import roc_curve
preds_for_label_true = np.random.normal(0.8, 0.5, 8).clip(0, 1)
preds_for_label_false = np.random.normal(0.2, 0.5, 992).clip(0, 1)
y_score = np.append(preds_for_label_true, preds_for_label_false)
fpr, tpr, thresholds = roc_curve(y, y_score=y_score)
plt.plot(fpr, tpr)
plt.title("ROC Curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```



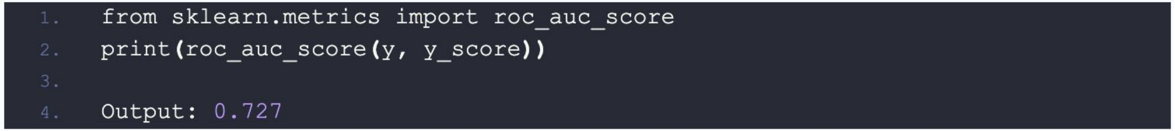
Precision Recall Curve

- This curve gives us direct information about different values we can achieve of precision and recall.
- However, it is important to note that precision is not necessarily monotonous with regard to the prediction threshold, even though generally precision increases as the threshold increases.
- Thus this graph can be a little more challenging to analyze in some cases.



AUC

- AUC, or area under the curve, is a popular metric that is used to summarize a graph by using a single number.
  - Usually, the curve referred to is the ROC curve, and thus the term is short for ROC AUC.
- AUC is also equal to the probability that our classifier will predict a higher score for a random positive example, than for a random negative example.



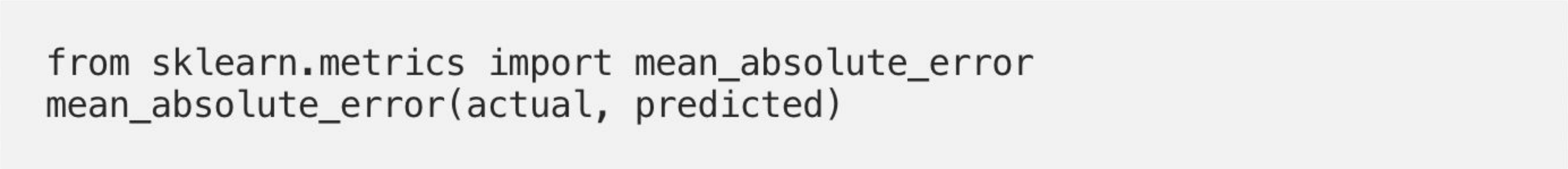
```
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y, y_score))

Output: 0.727
```

- AUC is a metric that is helpful in comparing different models since it summarizes the data from the whole ROC curve.

Mean Absolute Error (MAE)

- It is a measure of errors between paired observations expressing the same phenomenon.
- Examples of Y versus X include comparisons of predicted versus observed, subsequent time versus initial time, and one technique of measurement versus an alternative technique of measurement.
- It has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

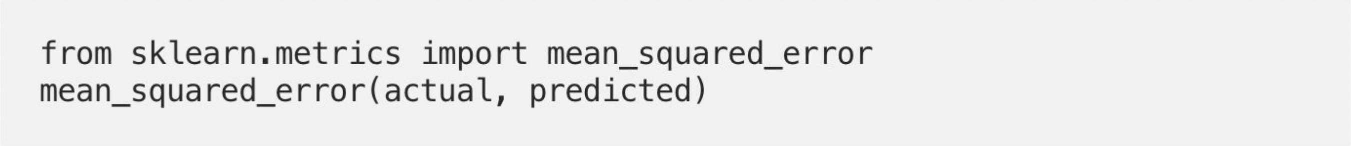


```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(actual, predicted)
```

Mean Square Error (MSE)

- It measures the average of the squares of the errors
    - that is, the average squared difference between the estimated values and the actual value.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

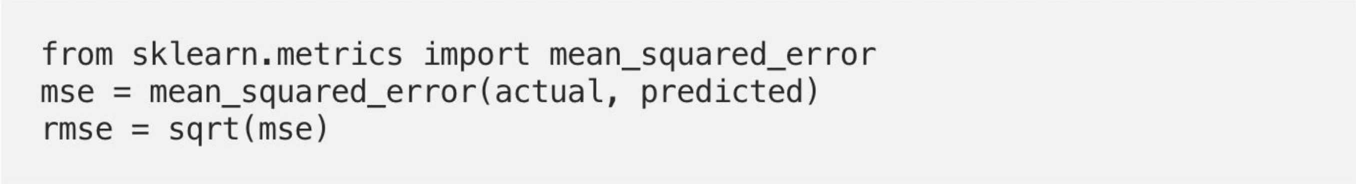


```
from sklearn.metrics import mean_squared_error
mean_squared_error(actual, predicted)
```

Root Mean Square Error (RMSE)

- In statistical modeling and particularly regression analyses, a common way of measuring the quality of the fit of the model is the RMSE (also called Root Mean Square Deviation)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{n}}$$



```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(actual, predicted)
rmse = sqrt(mse)
```

Never trust only a single metric

Always analyze different metrics to assess different aspects of the performance of a classifier/regressor!

The Theory of Learning

Learner's Input

- In the basic statistical learning setting, the learner has access to the following:
  - Domain set $\mathscr{X}$: Set of objects that we may wish to label.
  - Label set $\mathscr{Y}$. Set of possible labels.
    - We will restrict the label set to be a two-element set, usually {0,1} or {−1,+1}
  - Training data $S=((x_1,y_1)...(x_m,y_m))$ is a finite sequence of pairs in $\mathscr{X}\times\mathscr{Y}$
    - We call S the training examples or training set.

Learner’s Output

- The learner is requested to output a prediction rule, h: $\mathscr{X} \to \mathscr{Y}$.
- This function is also called a *predictor*, a *hypothesis*, or a *classifier*.
- The predictor can be used to predict the label of new domain points.
- We use the notation A(S) to denote the hypothesis that a learning algorithm, A, returns upon receiving the training sequence S.

Data-Generation Model

- We assume that the instances (the objects we have to classify) are generated by some probability distribution (in this case, representing the environment)
- Let us denote that probability distribution over $\mathscr{X}$ by $\mathscr{D}$.
  - It is important to note that we do not assume that the learner knows anything about this distribution.
  - For the type of learning tasks we discuss, this could be any arbitrary probability distribution
- As to the labels, in the current discussion we assume that there is some "correct" labeling function, f: $\mathscr{X} \rightarrow \mathscr{Y}$, and that $y_i = f(x_i)$ for all i.
  - The labeling function is unknown to the learner. In fact, this is just what the learner is trying to figure out.
- In summary, each pair in the training data S is generated by first sampling a point x_i according to $\mathscr{D}$ and then labeling it by f

Measures of success

- We define the error of a classifier to be the probability that it does not predict the correct label on a random data point generated by the aforementioned underlying distribution
  - the error of h is the probability to draw a random instance x, according to the distribution $\mathscr{D}$, such that h(x) does not equal f(x).
- Formally, given a domain subset $A \subset \mathscr{X}$, the probability distribution, $\mathscr{D}$, assigns a number, $\mathscr{D}(A)$, which determines how likely it is to observe a point $x \in A$.
  - In many cases, we refer to A as an event and express it using a function $\pi: \mathscr{X} \rightarrow \{0,1\}$, namely, $A = \{x \in \mathscr{X}: \pi(x)=1\}$.
  - In that case, we also use the notation $P_x \sim \mathscr{D}[\pi(x)]$ to express $\mathscr{D}(A)$.

Error of a Prediction Rule

- We define the error of a prediction rule, h: $\mathscr{X} \rightarrow \mathscr{Y}$, to be

$$L_{\mathcal{D},f}(h) \overset{\text{def}}{=} \underset{x\sim\mathcal{D}}{\mathbb{P}}[h(x) \neq f(x)] \overset{\text{def}}{=} \mathcal{D}(\{x : h(x) \neq f(x)\}).$$

  That is, the error of such h is the probability of randomly choosing an example x for which h(x) ≠ f(x).

  - The subscript ($\mathscr{D}$,f) indicates that the error is measured with respect to the probability distribution $\mathscr{D}$ and the correct labeling function f.

- $L_{\mathscr{D},f}$(h) has several names: *generalization error*, the *risk*, or the *true error of h*.

What does the learner know?

- The learner is blind to the underlying distribution $\mathscr{D}$ over the world and to the labeling function f.
- In our restaurant example, we have just arrived in a new city and we have no clue as to how waiting times are distributed and how to predict them.
- The only way the learner can interact with the environment is through observing the training set.

Empirical Risk Minimization

Goal of the algorithm A

- The goal of the learned algorithm A is to find h_S that minimizes the error with respect to the unknown $\mathscr{D}$ and f
  - the subscript S emphasizes the fact that the output predictor depends on S
- Since the learner does not know what $\mathscr{D}$ and f are, the true error is not directly available to the learner.
- As we have already seen we resort to minimize the training error

$$L_S(h) \overset{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}$$

  where [m] = {1, …, m}
- The terms *empirical error* and *empirical risk* are often used interchangeably for this error.

Empirical Risk Minimization

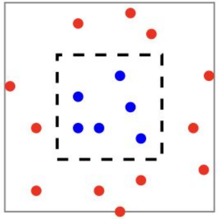
- Empirical risk:

$$L_S(h) \overset{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}$$

- Since the training sample is the snapshot of the world that is available to the learner, it makes sense to search for a solution that works well on that data
- This learning paradigm, coming up with a predictor h that minimizes $L_S(h)$, is called Empirical Risk Minimization or ERM for short.

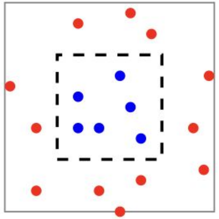
Overfitting (Again)

- Although the ERM rule seems very natural, without being careful, this approach may fail miserably.



- Assume that the probability distribution $\mathscr{D}$ is such that instances are distributed uniformly within the gray square and the labeling function, f, determines the label to be 1 if the instance is within the inner blue square, and 0 otherwise.
  - The area of the gray square in the picture is 2 and the area of the blue square is 1.

Overfitting (Again)



- Consider the following predictor:

$$h_S(x) = \begin{cases} y_i & \text{if } \exists i \in [m] \text{ s.t. } x_i = x \\ 0 & \text{otherwise.} \end{cases}$$

- Clearly, no matter what the sample is, $L_S(h_S) = 0$
  - it is one of the empirical-minimum-cost hypotheses;
  - no classifier can have smaller error.
- On the other hand, the true error of any classifier that predicts the label 1 only on a finite number of instances is, in this case, 1/2. Thus, $L_{\mathscr{D}}(h_S) = 1/2$.
  - We have found a predictor whose performance on the training set is excellent, yet its performance on the true "world" is very poor.

Overfitting (Again)

Intuitively, overfitting occurs when our hypothesis fits the training data “**too well**”

(perhaps like the everyday experience that a person who provides a perfect detailed explanation for each of his single actions may raise suspicion)

Empirical Risk Minimization and Inductive Bias

- We will search for conditions under which there is a guarantee that ERM does not overfit
    - conditions under which when the ERM predictor has good performance with respect to the training data, it is also highly likely to perform well over the underlying data distribution.
- A common solution is to apply the ERM learning rule over a restricted search space.

Inductive Bias

- Formally, the learner should choose in advance (before seeing the data) a set of predictors. This set is called a hypothesis class and is denoted by $\mathcal{H}$. Each h $\in \mathcal{H}$ is a function mapping from $\mathcal{X}$ to $\mathcal{Y}$.
- For a given class $\mathcal{H}$, and a training sample, S, the ERM$_{\mathcal{H}}$ learner uses the ERM rule to choose a predictor h $\in \mathcal{H}$, with the lowest possible error over S. Formally

$$\mathrm{ERM}_{\mathcal{H}}(S) \in \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, L_S(h),$$

- By restricting the learner to choosing a predictor from $\mathcal{H}$, we bias it toward a particular set of predictors.
- Such restrictions are often called an inductive bias.

Finite Hypothesis Space

Finite Hypothesis Classes

- The simplest type of restriction on a class is imposing an upper bound on its size (that is, the number of predictors h in $\mathcal{H}$).
- If $\mathcal{H}$ is a finite class then $\mathrm{ERM}_{\mathcal{H}}$ will not overfit
  - provided it is based on a sufficiently large training sample (this size requirement will depend on the size of $\mathcal{H}$).
- Limiting the learner to prediction rules within some finite hypothesis class may be considered as a reasonably mild restriction.
  - For example, $\mathcal{H}$ can be the set of all predictors that can be implemented by a C++ program written in at most 109 bits of code.
  - In our previous example, we used the class of axis aligned rectangles. While this is an infinite class
  - if we discretize the representation of real numbers, say, by using a 64 bits floating-point representation, the hypothesis class becomes a finite class.

The Realizability Assumption

- For a training sample, S, labeled according to some f: $\mathcal{X} \to \mathcal{Y}$, let h_S denote a result of applying $ERM_{\mathcal{H}}$ to S, namely

$$h_S \in \underset{h \in \mathcal{H}}{\operatorname{argmin}} L_S(h)$$

Definition [The Realizability Assumption]. There exists $h^* \in \mathcal{H}$ s.t. $L_{(\mathcal{D},f)}(h^*) = 0$.

This implies that with probability 1 over random samples, S, where the instances of S are sampled according to $\mathcal{D}$ and are labeled by f, we have $L_S(h^*) = 0$.

SAPIENZA
UNIVERSITÀ DI ROMA

The IID hypothesis

- Clearly, any guarantee on the error with respect to the underlying distribution, $\mathscr{D}$, for an algorithm that has access only to a sample S should depend on the relationship between $\mathscr{D}$ and S.

The i.i.d. assumption: The examples in the training set are *independently and identically distributed* (i.i.d.) according to the distribution $\mathscr{D}$. That is, every x_i in S is freshly sampled according to $\mathscr{D}$ and then labeled according to the labeling function, f.

We denote this assumption by S ~ $\mathscr{D}_m$ where m is the size of S, and $\mathscr{D}_m$ denotes the probability over m-tuples induced by applying $\mathscr{D}$ to pick each element of the tuple independently of the other members of the tuple.

$L_{(\mathscr{D},f)}$ as a Random Variable

- Since $L_{(\mathscr{D},f)}(h_S)$ depends on the training set, S, and that training set is picked by a random process, there is randomness in the choice of the predictor h_S and, consequently, in the risk $L_{(\mathscr{D},f)}(h_S)$.
- It is not realistic to expect that *with full certainty* S will suffice to direct the learner toward a good classifier (from the point of view of $\mathscr{D}$)
  - there is always some probability that the sampled training data happens to be *very nonrepresentative* of the underlying $\mathscr{D}$.
- Usually, we denote the probability of getting a nonrepresentative sample by δ, and call (1 − δ) the confidence parameter of our prediction.
- since we cannot guarantee perfect label prediction, we introduce another parameter for the quality of prediction, the accuracy parameter, ε
  - We interpret the event $L_{(\mathscr{D},f)}(h_S) > \varepsilon$ as a failure of the learner, while if $L_{(\mathscr{D},f)}(h_S) \leq \varepsilon$ we view the output of the algorithm as an **approximately correct** predictor.

Bounding the Probability of Making Mistakes

- Let f: $\mathscr{X} \to \mathscr{Y}$ be a labelling function
- Let $S|_x = (x_1, \ldots, x_m)$ be the instances of the training set. We would like to upper bound

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\})$$

the probability to sample m-tuple of instances that will lead to failure of the learner.

Bounding the Probability of Making Mistakes

- Let $\mathcal{H}_B$ be the set of bad hypotheses

$$\mathcal{H}_B = \{h \in \mathcal{H} : L_{(\mathcal{D},f)}(h) > \epsilon\}$$

- In addition, let

$$M = \{S|_x : \exists h \in \mathcal{H}_B, L_S(h) = 0\}$$

  be the set of misleading samples.
  - Namely, for every $S|_x \in M$, there is a “bad” hypothesis, $h \in \mathcal{H}_B$, that looks like a “good” hypothesis on $S|_x$.

Bounding the Probability of Making Mistakes

- We seek to bound the probability of the event $L_{(\mathcal{D},f)}(h_S) > \varepsilon$
- The realizability assumption implies that $L_S(h_S) = 0$
- It follows that $L_{(\mathcal{D},f)}(h_S) > \varepsilon$ can only happen if for $h \in \mathcal{H}_B$ we have $L_S(h) = 0$
  - In other words, this event will only happen if *our sample is in the set of misleading samples, M*
- Formally we have shown that

$$\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\} \subseteq M$$

Note that we can rewrite M as

$$M = \bigcup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\}$$

Hence,

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\}) \ \leq \ \mathcal{D}^m(M) = \mathcal{D}^m(\cup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\})$$

Interlude: Union Bound

- Lemma [Union Bound]. For any two sets A, B and a distribution $\mathscr{D}$ we have that $\mathscr{D}(A \cup B) \leq \mathscr{D}(A) + \mathscr{D}(B)$.

Proof (trivial)

$$\mathscr{D}(A \cup B) = \mathscr{D}(A) + \mathscr{D}(B) - \mathscr{D}(A \cap B) \leq \mathscr{D}(A) + \mathscr{D}(B).$$

Bounding the Probability of Making Mistakes

- Applying the union bound to the right-hand side of

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\}) \;\leq\; \mathcal{D}^m(M) = \mathcal{D}^m(\cup_{h \in \mathcal{H}_B} \{S|_x : L_S(h) = 0\})$$

  we get

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\}) \;\leq\; \sum_{h \in \mathcal{H}_B} \mathcal{D}^m(\{S|_x : L_S(h) = 0\})$$

- and by observing that $L_S(h) = 0$ is equivalent to $\forall i, h(x_i) = f(x_i)$ we get that

$$\begin{aligned}\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) &= \mathcal{D}^m(\{S|_x : \forall i, h(x_i) = f(x_i)\}) \\ &= \prod_{i=1}^{m} \mathcal{D}(\{x_i : h(x_i) = f(x_i)\}).\end{aligned}$$

Bounding the Probability of Making Mistakes

- Applying the union bound to the right-hand side of

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\}) \;\le\; \mathcal{D}^m(M) = \mathcal{D}^m(\cup_{h\in\mathcal{H}_B}\{S|_x : L_S(h) = 0\})$$

we get

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\}) \;\le\; \sum_{h\in\mathcal{H}_B} \mathcal{D}^m(\{S|_x : L_S(h) = 0\})$$

and by observing that $L_S(h) = 0$ is equivalent to $\forall i, h(x_i) = f(x_i)$ we get that

$$\begin{aligned}\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) &= \mathcal{D}^m(\{S|_x : \forall i, h(x_i) = f(x_i)\}) \\ &= \prod_{i=1}^{m} \mathcal{D}(\{x_i : h(x_i) = f(x_i)\}).\end{aligned}$$

$$\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) = \mathcal{D}^m(\{S|_x : \forall i, h(x_i) = f(x_i)\}) \\ = \prod_{i=1}^{m} \mathcal{D}(\{x_i : h(x_i) = f(x_i)\}).$$

Bounding the Probability of Making Mistakes

- For each individual sampling of an element of the training set we have

$$\mathcal{D}(\{x_i : h(x_i) = y_i\}) = 1 - L_{(\mathcal{D},f)}(h) \leq 1 - \epsilon,$$

  where the last inequality follows from the fact that h ∈ $\mathcal{H}_B$
- Combining the previous equation with Equation (*) and using the inequality $1 - \varepsilon \leq e^{-\varepsilon}$ we obtain that for every h ∈ $\mathcal{H}_B$,

$$\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) \leq (1 - \epsilon)^m \leq e^{-\epsilon m}$$

Bounding the Probability of Making Mistakes

- Combining

$$\begin{aligned}\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) &= \mathcal{D}^m(\{S|_x : \forall i, h(x_i) = f(x_i)\}) \\ &= \prod_{i=1}^{m} \mathcal{D}(\{x_i : h(x_i) = f(x_i)\}).\end{aligned}$$

and

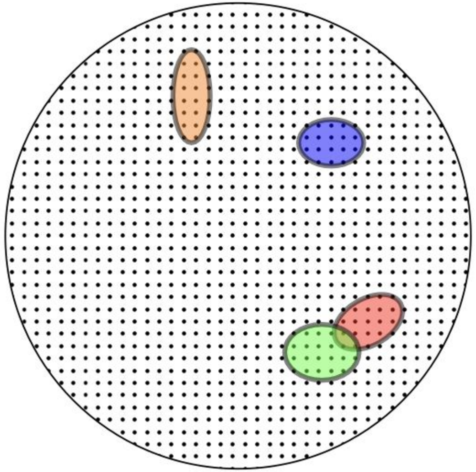
$$\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) \leq (1-\epsilon)^m \leq e^{-\epsilon m} \qquad (*)$$

we can conclude that

$$\mathcal{D}^m(\{S|_x : L_{(\mathcal{D},f)}(h_S) > \epsilon\}) \;\leq\; |\mathcal{H}_B|\,e^{-\epsilon m} \;\leq\; |\mathcal{H}|\,e^{-\epsilon\, m}$$

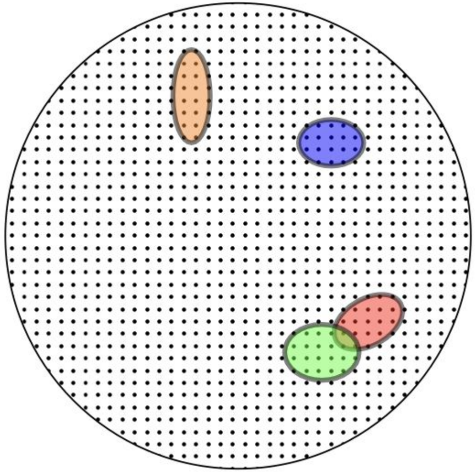
This because the above inequality (*) holds for every $h \in \mathcal{H}_B$

A Picture is Worth a Thousand Words



Each point in the large circle represents a possible m-tuple of instances. Each colored oval represents the set of “misleading” m-tuple of instances for some “bad” predictor $h \in \mathcal{H}_B$. The ERM can potentially overfit whenever it gets a misleading training set S . That is, for some $h \in \mathcal{H}_B$ we have $L_S(h) = 0$.

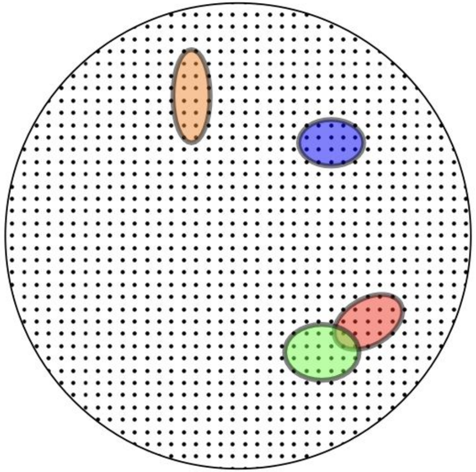
A Picture is Worth a Thousand Words



$$\mathcal{D}^m(\{S|_x : L_S(h) = 0\}) \leq (1-\epsilon)^m \leq e^{-\epsilon m}$$

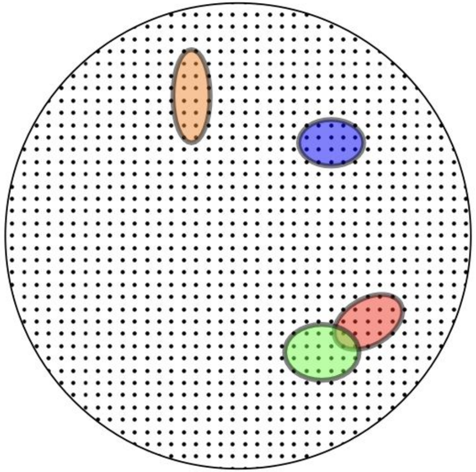
The equation above guarantees that for each individual bad hypothesis, h ∈ $\mathscr{H}_B$, at most $(1 − ε)^m$-fraction of the training sets would be misleading. In particular, the larger m is, the smaller each of these colored ovals becomes.

A Picture is Worth a Thousand Words



The union bound formalizes the fact that the area representing the training sets that are misleading with respect to some $h \in \mathcal{H}_B$ (that is, the training sets in M) is at most the sum of the areas of the colored ovals.

A Picture is Worth a Thousand Words



Therefore, it is bounded by $|\mathcal{H}_B|$ times the maximum size of a colored oval. Any sample S outside the colored ovals cannot cause the ERM rule to overfit.

Corollary

Let $\mathscr{H}$ be a finite hypothesis class. Let δ ∈ (0,1) and ε > 0 and let m be an integer that satisfies

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}$$

Then, for any labeling function, f, and for any distribution, $\mathscr{D}$, for which the realizability assumption holds (that is, for some h ∈ $\mathscr{H}$, $L_{(\mathscr{D},f)}(h) = 0$), with probability of at least 1 − δ over the choice of an i.i.d. sample S of size m, we have that for every ERM hypothesis, h_S, it holds that $L_{(\mathscr{D},f)}(h_S) \leq \varepsilon$.

The preceding corollary tells us that for a sufficiently large m, the $\mathrm{ERM}_{\mathscr{H}}$ rule over a finite hypothesis class will be **probably** (with confidence 1−δ) **approximately** (up to an error of ε) **correct**.

Probably Approximately Correct (PAC)

- The underlying principle is that any hypothesis that is seriously wrong will almost certainly be “found out” with high probability after a small number of examples, because it will make an incorrect prediction.
- Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: that is, it must be **probably approximately correct** (**PAC**).
- In the previous case of finite hypothesis space, sufficiently large means:

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\epsilon}$$

Another Simple PAC Theorem: Boolean Functions

- $|\mathscr{H}| = 2^{2^n}$ ⇒ finite hypothesis space.
- For boolean functions the 0/1 loss is appropriate
- The error rate of a hypothesis h is defined formally as the expected generalization error for examples drawn from the stationary distribution:

$$\text{error}(h) = GenLoss_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x))\, P(x,y)$$

  here we use P instead of $\mathscr{D}$ to denote the distribution, and
  $L_{0/1}(y, \hat{y}) = 0$ if $y = \hat{y}$, else 1.

- In other words, error(h) is the probability that h misclassifies a new example.

Another Simple PAC Theorem: Boolean Functions

- Let us derive a bound on the probability that a “seriously wrong” hypothesis $h_b \in \mathscr{H}_{bad}$ is consistent with the first N examples.
- $\text{error}(h_b) > \varepsilon \Rightarrow$ probability that it agrees with a given example is at most $1 - \varepsilon$
- Since the examples are independent, the bound for N examples is:
  $P(h_b \text{ agrees with N examples}) \leq (1 - \varepsilon)^N$
- The probability that $\mathscr{H}_{bad}$ contains at least one consistent hypothesis is bounded by the sum of the individual probabilities:
  $P(\mathscr{H}_{bad} \text{ contains a consistent hypothesis}) \leq |\mathscr{H}_{bad}|(1 - \varepsilon)^N \leq |\mathscr{H}|(1 - \varepsilon)^N$
- We would like to reduce the probability of this event below some small number δ

Another Simple PAC Theorem: Boolean Functions

- P($\mathcal{H}_{\text{bad}}$ contains a consistent hypothesis) ≤ $|\mathcal{H}|(1 - \varepsilon)^N \leq \delta$
- Again, let us use the fact that $1 - \varepsilon \leq e^{-\varepsilon}$ to get

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|\mathcal{H}|\right)$$

- Thus, with probability at least $1 - \delta$, after seeing this many examples, the learning algorithm will return a hypothesis that has error at most ε. In other words, it is probably approximately correct.
- The number of required examples, as a function of ε and δ, is called the sample complexity of the learning algorithm.

Sample Complexity of Boolean Hypothesis

- $|\mathscr{H}| = 2^{2^n}$ ⇒ finite hypothesis space ⇒ sample complexity grows exponentially ⇒ PAC-learning requires to explore nearly all possibilities!!!
  - $\mathscr{H}$ contains enough hypotheses to classify any given set of examples in all possible ways.
  - In particular, for any set of N examples, the set of hypotheses consistent with those examples contains equal numbers of hypotheses that predict x_{N+1} to be positive and hypotheses that predict x_{N+1} to be negative
- To obtain real generalization to unseen examples, then, it seems we need to restrict the hypothesis space $\mathscr{H}$ in some way
  - if we do restrict the space, we might eliminate the true function altogether.
- There are three ways to escape this dilemma.
  - We have seen two of them already

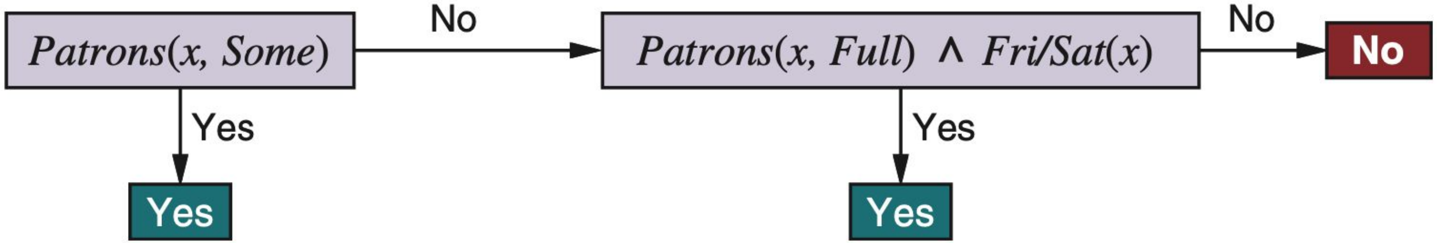
Approaching the Complexity

- Focus on learnable subsets of the entire hypothesis space of Boolean functions
- Assumption ⇐ *the restricted hypothesis space contains a hypothesis h that is close enough to the true function f*
  - the benefits are that the restricted hypothesis space allows for effective generalization and is typically easier to search

PAC Learning: Learning Decision Lists

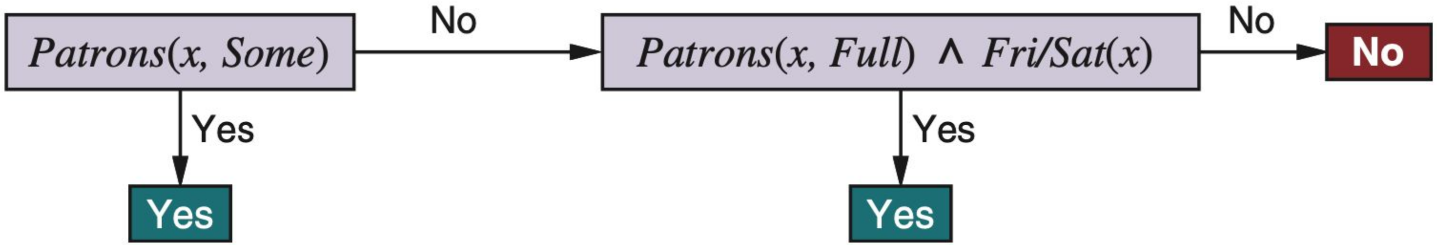
Decision Lists

- A decision list consists of a series of tests, each of which is a conjunction of literals.
- If a test succeeds when applied to an example description, the decision list specifies the value to be returned.
- If the test fails, processing continues with the next test in the list.



Decision Lists

- if we restrict the size of each test to at most k literals, then it is possible for the learning algorithm to generalize successfully from a small number of examples.
- k-DL is a decision list with up to k conjunctions.
- k-DL(n) to denote a k-DL using n Boolean attributes
- This example is in 2-DL(3).



k-DL is Learnable

- Any function in k-DL can be approximated accurately after training on a reasonable number of examples
- Let us compute the number of possible hypotheses
  - Conj(n,k) ⇐ set of conjunctions of at most k literals using n attributes
  - Decision lists are constructed from tests, and each test can be attached to either a Yes or a No outcome or can be absent from the decision list ⇒ there are at most $3^{|Conj(n,k)|}$ distinct sets of component tests.
  - Each of these sets of tests can be in any order
- $|k\text{-DL}(n)| \leq 3^c c!$ where c = |Conj(n,k)|
- And $|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)$
- Hence $|k\text{-DL}(n)| = 2^{O(n^k \log 2(n^k))}$

k-DL is Learnable

- Therefore, combining

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)$$

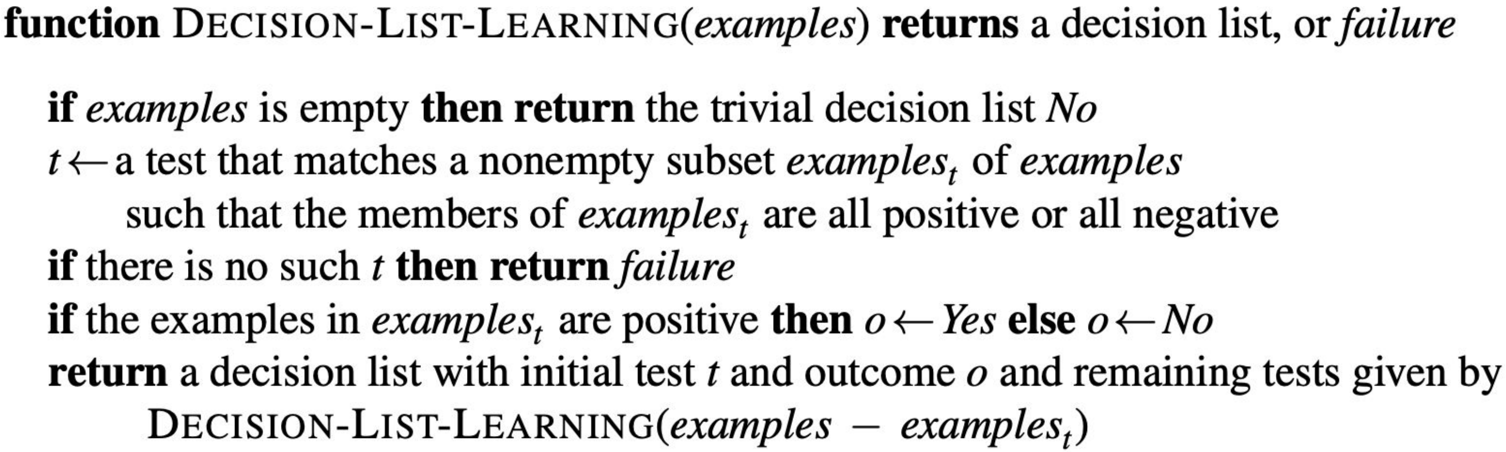
with

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|\mathcal{H}|\right)$$

- We can show that the number of examples needed for PAC learning a k-DL(n) function is polynomial in n:

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)$$

Learning Decision Lists



```
function DECISION-LIST-LEARNING(examples) returns a decision list, or failure

  if examples is empty then return the trivial decision list No
  t ← a test that matches a nonempty subset examples_t of examples
        such that the members of examples_t are all positive or all negative
  if there is no such t then return failure
  if the examples in examples_t are positive then o ← Yes else o ← No
  return a decision list with initial test t and outcome o and remaining tests given by
          DECISION-LIST-LEARNING(examples − examples_t)
```

Decision Lists vs Decision Trees

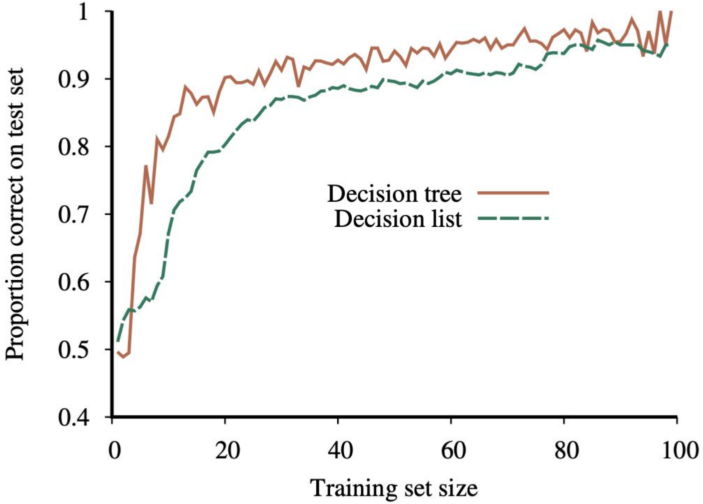


Figure 19.12 Learning curve for Decision-List-Learning algorithm on the restaurant data. The curve for Learn-Decision-Tree is shown for comparison; decision trees do slightly better on this particular problem.

A More General Learning Model

Removing the Realizability Assumption

- Learning models in this class are called Agnostic PAC.
- Recall that the realizability assumption requires that there exists $h^* \in \mathcal{H}$ such that $P_{x \sim \mathcal{D}}[h^*(x) = f(x)] = 1$
  - This is not always the case in reality (for example you have a partial view of the real world, two restaurants with the same features might have different waiting times due to other factors, e.g., the cook has gotten a phone call)
- Thus, from now on, let $\mathcal{D}$ be a probability distribution over $\mathcal{X} \times \mathcal{Y}$, where, as before, $\mathcal{X}$ is our domain set and $\mathcal{Y}$ is a set of labels ({0, 1} for binary classification).
  - $\mathcal{D}$ is a joint distribution over domain points and labels.
  - Two parts: a distribution $\mathcal{D}_x$ over unlabeled domain points (sometimes called the marginal distribution) and a conditional probability over labels for each domain point, D((x,y)|x)

The empirical and the True Error Revised

- For a probability distribution, $\mathscr{D}$, over $\mathscr{X} \times \mathscr{Y}$, how likely if h to make an error when labeled points are randomly drawn according to $\mathscr{D}$?
- We redefine the true error (or risk) of a prediction rule h to be

$$L_{\mathcal{D}}(h) \overset{\text{def}}{=} \underset{(x,y)\sim\mathcal{D}}{\mathbb{P}}[h(x) \neq y] \overset{\text{def}}{=} \mathcal{D}(\{(x,y) : h(x) \neq y\})$$

- We would like to find a predictor, h, for which that error will be minimized. However, the learner does not know the data generating $\mathscr{D}$.
- What the learner does have access to is the training data, S.

SAPIENZA
UNIVERSITÀ DI ROMA

The empirical and the True Error Revised

- The definition of the empirical risk remains the same

$$L_S(h) \overset{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}.$$

- Given S, a learner can compute $L_S(h)$ for any function h: $\mathscr{X} \rightarrow \{0,1\}$.
    - Note that $L_S(h) = L_{\mathscr{D}(\text{uniform over S})}(h)$

The Goal

- We seek to find some hypothesis, h: $\mathscr{X} \rightarrow \mathscr{Y}$, that (probably approximately) minimizes the true risk, $L_{\mathscr{D}}$(h)

$$L_{\mathcal{D}}(h) \overset{\text{def}}{=} \underset{(x,y)\sim\mathcal{D}}{\mathbb{P}}[h(x) \neq y] \overset{\text{def}}{=} \mathcal{D}(\{(x,y) : h(x) \neq y\})$$

Definition – Agnostic PAC Learnability

- A hypothesis class $\mathscr{H}$ is agnostic PAC learnable if there exist a function $m_{\mathscr{H}}:(0,1)^2 \rightarrow \mathbb{N}$ and a learning algorithm with the following property:
  - For every ε, δ ∈ (0,1) and for every distribution $\mathscr{D}$ over $\mathscr{X} \times \mathscr{Y}$, when running the learning algorithm on $m \geq m_{\mathscr{H}}(\varepsilon, \delta)$ i.i.d. examples generated by $\mathscr{D}$, the algorithm returns a hypothesis h such that, with probability of at least 1 − δ (over the choice of the m training examples)

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon.$$

Agnostic PAC Learnability Generalizes PAC Learning

- If the realizability assumption holds, agnostic PAC learning provides the same guarantee as PAC learning.
- In that sense, Agnostic PAC learning generalizes the definition of PAC learning.
  - When the realizability assumption does not hold, no learner can guarantee an arbitrarily small error.
- Nevertheless, under the definition of agnostic PAC learning, a learner can still declare success if its error is not much larger than the best error achievable by a predictor from the class $\mathcal{H}$.
  - This is in contrast to PAC learning, in which the learner is required to achieve a small error in absolute terms and not relative to the best error achievable by the hypothesis class.

Fondamenti di IA

End of Lecture

02 - Model Selection and Theory of Learning

SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri