CPSC 483
Evaluation of ML models

Anand Panangadan

apanangadan@fullerton.edu

What we will cover today

- Evaluation
  - Regression
  - Classification

Why evaluation?

- Quantitatively compare different models for a given dataset
  - Which one is better?
- Identify optimal values for hyperparameters
  - Hyperparameters of the ML model
  - Hyperparameters of pre-processing steps

Supervised Learning

Features

Labels

Petal Length	Petal Width	Sepal Length	Sepal Width	Species
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
6.2	2.9	4.3	1.3	Versicolor
6.3	3.3	6.0	2.5	Virginica
5.1	2.5	3.0	1.1	Versicolor
7.1	3.0	5.9	2.1	Virginica

Training data

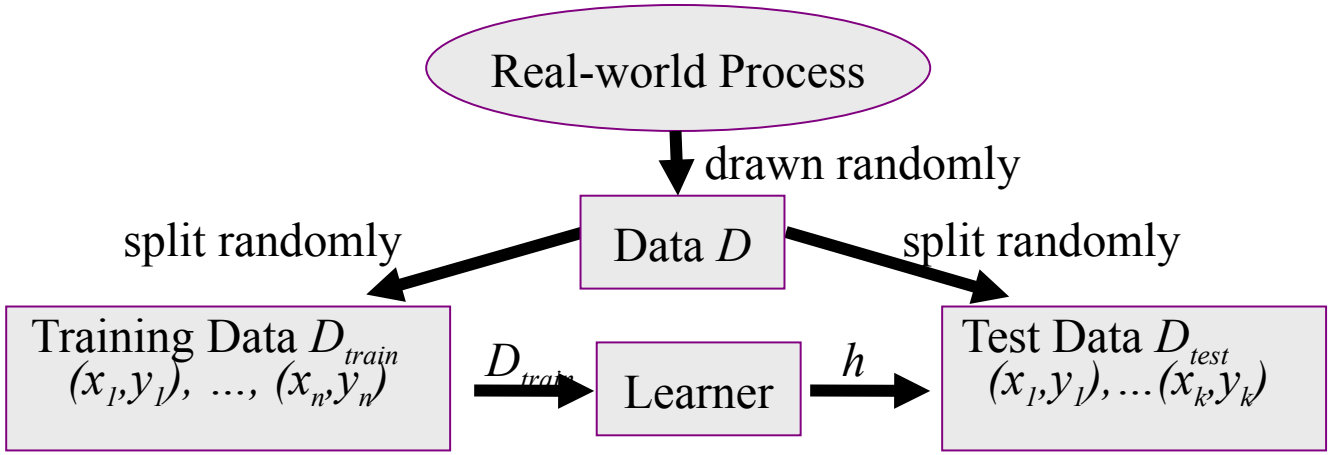
5.1	3.5	1.4	0.2	Setosa	?
7.1	3.0	5.9	2.1	Virginica	?

Test data

Ground truth

Predictions

Test/Training Split



Overfitting

- Some attributes are irrelevant to the decision-making process
- But the decision tree learning algorithm does not know which
- Example:
  - Problem of trying to predict the roll of a die
  - The experiment data includes
    - Color
    - Day of week
- All are irrelevant to its outcome but the ID3 algorithm will use these to differentiate examples

Overfitting

- If the hypothesis space has many dimensions because of a large number of attributes:
- There will be *meaningless* regularity in the data just from random chance
- Irrelevant to the true, important, distinguishing features

Overfitting

- Overfitting: fitting the model (decision tree) to the **training set** "too well"
- Performance on the test set degrades.
- Example of overfitting risk parameter for wait-at-restaurant decision?
  - Using restaurant name.

Overfitting

- **D: the entire distribution of data**
- **T: the training set**
- **Hypothesis (decision tree) h ∈ H overfits D if**

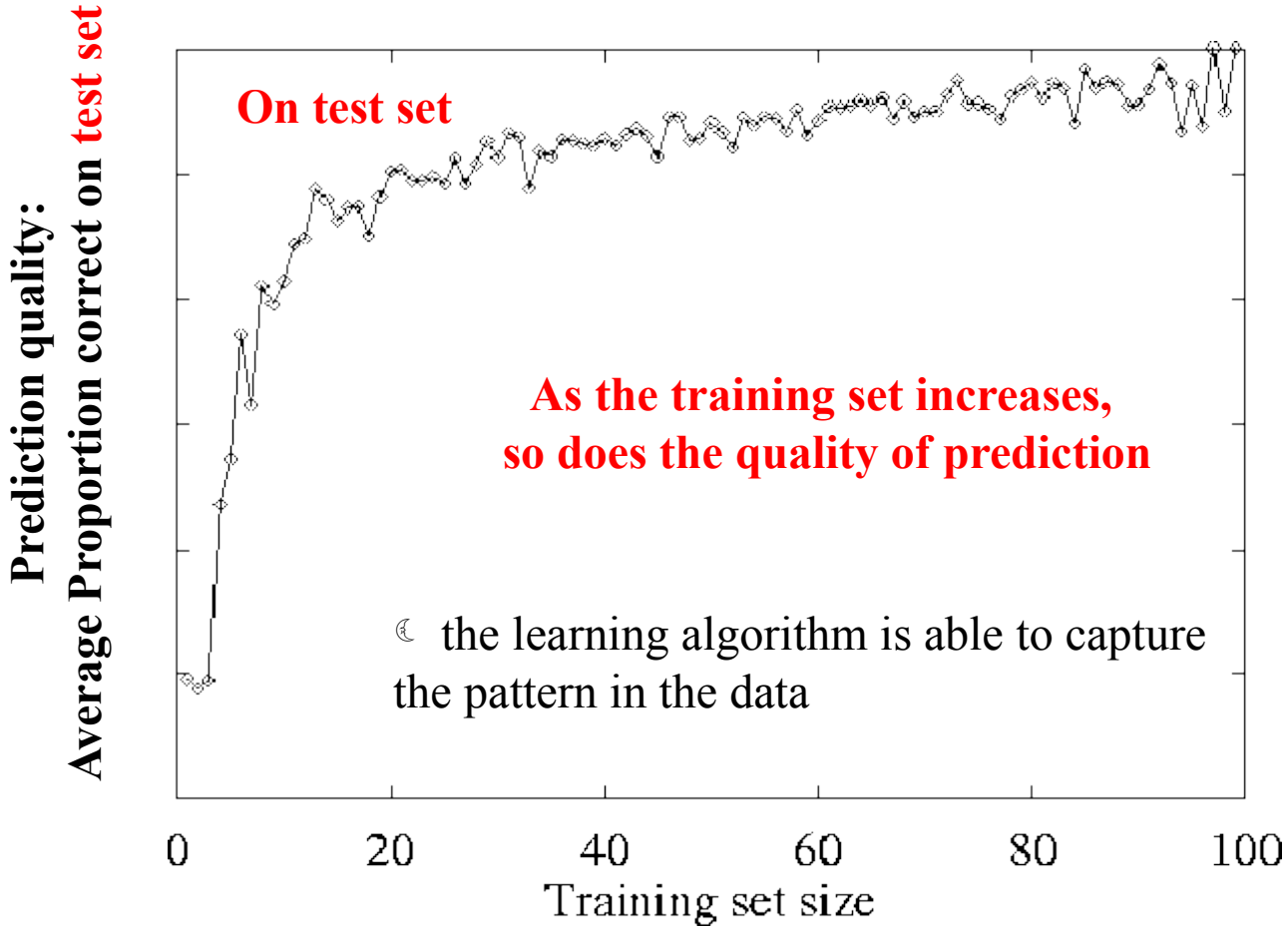
∃ h’≠ h ∈ H such that

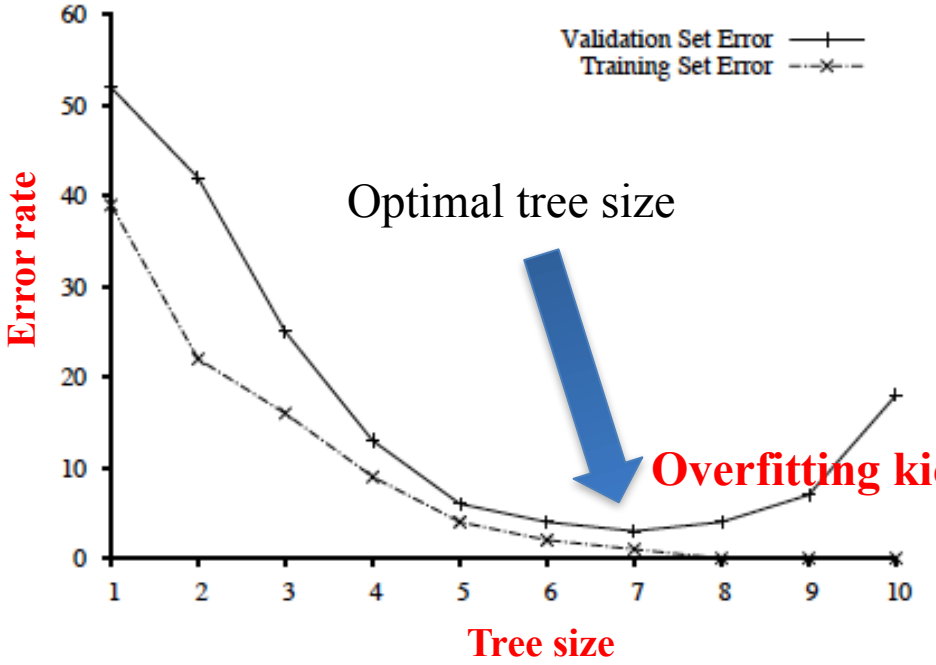
$error_T(h) < error_T(h')$ but

$error_D(h) > error_D(h')$

Note: can only estimate error on full distribution by using a *test* data set

Learning Curve





Optimal tree size

Tree size as
a parameter in our
DT learning alg.

Overfitting kicks in…

With larger and larger trees,
we just do better and better on the training set!

But performance on the validation set decreases

Reasons for watching out for over-fitting

- Many kinds of "noise" that could occur in the examples:
  - Two examples have same attribute/value pairs, but different classifications
    - report majority classification for the examples corresponding to the node deterministic hypothesis.
    - report estimated probabilities of each classification using the relative frequency (if considering stochastic hypotheses)
  - Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  - The classification is wrong (e.g., + instead of -) because of some error

How to choose test data: Holdout

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation

- **Random sampling**
  - a variation of holdout
  - Repeat holdout *n* times
  - accuracy = average of the *n* accuracies

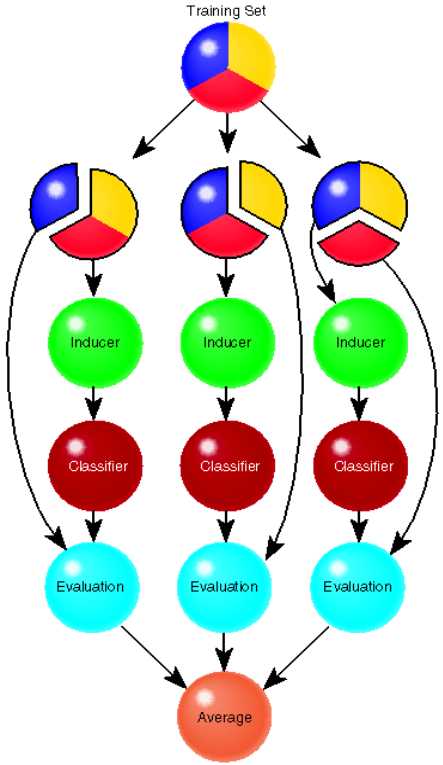
Holdout Validation

1. Randomly divide data into two disjoint sets: **training set** and **test set**.
   - "No peeking"!
2. Apply learning algorithm to training set generating tree *h*
3. Measure performance of *h* w.r.t. test set
   - measures generalization to unseen data

How to choose test data: Cross-Validation

- **K-fold Cross-validation**
  - Randomly partition the data into *k* *mutually exclusive* subsets, each approximately equal size
  - Do *k* iterations:
    - At *i*-th iteration, use D_i as test set and others as training set
- More efficient use of data than "holdout cross-validation"

3-fold cross-valiadtion



How to choose test data: Leave-one-out

- Extreme case of *k*-fold cross-validation
- Put all but one sample in training data
  - Keep remaining for test
  - Repeat for all *n* choices
  - *n*: number of data points

Evaluation of Regression

- Can use the performance measures used for training
- Root mean square error (RMSE)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\Big)^2}$$

- Mean absolute error (MAE)

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m} h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\Big|$$

Scikit-learn's cross validation

```
from sklearn.model_selection import
cross_val_score
…

model_rmses = -cross_val_score(mymodel,
     train_X, train_labels,
     scoring="neg_root_mean_squared_error",
     cv=10)
```

Identifying best model hyperparameters during evaluation

- Models can have hyperparameters
  - E.g., max number of input features
  - Is specific to the type of model
- Can repeat evaluation step for different values of hyperparameters
- Select value that gives best results
- Note: pre-processing step hyperparameters can also be considered here

Identifying best model hyperparameters

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([ ("preprocessing",
preprocessing),
                ("random_forest",
RandomForestRegressor()) ])

param_grid = [{'random_forest__max_features': [4, 6,
8] }]
        # list of dictionaries
grid_search = GridSearchCV(full_pipeline, param_grid,
cv=3,
        scoring='neg_root_mean_squared_error')

grid_search.fit(housing, housing_labels)
grid_search.best_params_
```

Class work

- Create a copy of the `02_end_to_end_machine_learning_project` Python notebook
- Retain only the following steps (i.e., delete the others):
  - Keep aside a test dataset that is 20% of the full dataset using random sampling
  - Replace missing values with median values
  - Replace categorical features using one-hot-encoding
  - Compute a few transformed variables (ratios of two other variables)
  - Standardize all numerical variables
- Create one pipeline with
  - Above pre-processing steps
  - RandomForestRegressor
- Perform 3-fold cross-validation and identify best `max_features` parameter for RandomForestRegressor

Textbook code

- Textbook code
- Textbook code on Google Colab
- Open

  `02_end_to_end_machine_learning_project.ipynb`

Model selection

- After best model size is found from the error rate curve on validation data, re-train on *all* training data to get final model for deployment.
- Evaluate final model on the test data (not used before) to estimate true generalization error (to unseen examples).

Classifier Evaluation Metrics: Confusion Matrix

Compare ground truth with predictions

- If they match it is called "True"
  - If not, it is "False"
- A prediction can take one of two values
  - called Positive or Negative
- Also called "Contingency Table"
  - In R: `table(x, y)`

	Predicted class	
Actual class	C_1(Positives)	¬ C_1(Negatives)
C_1(Positives)	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1(Negatives)	**False Positives (FP)**	**True Negatives (TN)**

Accuracy and Error Rate

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
			All

- **Accuracy**: percentage of test set data that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

  Note: P = TP + FN

  N = FP + TN

  All = P +N = TP + FN + FP + TN

Be careful of “Accuracy”

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive

Accuracy is dominated by the larger set (of positives or negatives) and favors trivial classifiers.

e.g. if 5% of items are truly positive, then a classifier that always says “negative” is 95% accurate.

Dummy classifier

```
from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train)

print(any(dummy_clf.predict(X_train)))
# prints False for ALL cases where there are more Negative
instances
```

Precision and Recall

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive?

Precision =

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

Recall =

- Perfect score is 1.0
- Inverse relationship between precision & recall

F-measures

- ***F* measure (*F_1* or *F*-score)**:
- harmonic mean of precision and recall

F =

Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
			All

- **Sensitivity**: True Positive rate
  - **Sensitivity = TP/P**
  - **= TP/(TP + FN) = Recall**
- **Specificity**: True Negative rate
  - **Specificity = TN/N**

Classwork

Actual Class\Predicted class	cancer = yes	cancer = no
cancer = yes	**90**	**210**
cancer = no	**140**	**9560**

- **Calculate**
  - **Accuracy**
  - **Error**
  - **Precision of cancer=yes**
  - **Recall of cancer=yes**
  - **F-score of cancer=yes**

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*)
cancer = no	**140**	**9560**	9700	98.56 (*specificity*)
Total	230	9770	10000	96.50 (*accuracy*)

Precision = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Classwork

- **Calculate**
  - Accuracy
  - Error rate
  - Precision
  - Recall
  - F1-score
- "Positive" class is High Risk

Credit score	Income ($)	Risk
600	50,000	High
650	60,000	Low
800	55,000	Low
550	55,000	Low
660	50,000	High
750	58,000	Low

- For the following two classifiers
  1. Risk is always High
  2. Only people with Credit score > 700 OR Income >= $60,000 have Low Risk. All others have High Risk

In two steps:

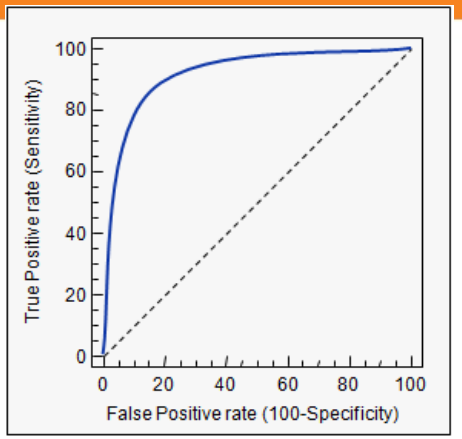
```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(mymodel, X_train, y_train,
cv=3)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_train_pred)
cm
```

Class work

- Modify the Python notebook:
  - `03_classification.ipynb`

- Train a SGDClassifier to identify which of the digits in the MNIST dataset is ODD (1, 3, 5, 7, 9)
- Show the confusion matrix of the classifier

Visualizing the True and False Positive Rates



- **Calculate TP and FP scores**
  - By evaluating the classifier (that returns a probability score 0-1) at various threshold values
  - Count TP and FP, and plot the scores
- **The Receiver Operating Characteristic (ROC) curve**
  - A graphical approach for displaying the **tradeoff** between true positive rate and false positive rate.
  - Closely related to precision
  - A good classifier should be located as close as possible to the upper-left corner, while a random classifier should reside along the main diagonal.
    - Useful for comparing the relative performance among different classifiers.
    - The **area** under the ROC curve provides information on which classifier is better on average.
    - If the model is perfect, its area would be 1, random guess = 0.5.

Acknowledgement

- Content based on “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow,” Aurélien Géron, 3rd Edition (October 2022), O'Reilly Media, Inc.