Evaluation and Methodology

CSCI-P556 Applied Machine Learning

Lecture 6

D.S. Williamson

Agenda and Learning Outcomes

Today's Topics

- **Topics**:
  - Handling Categorical data (e.g. Data cleaning)
  - Feature scaling
  - Measures of performance for classification
  - N-Fold Cross Validation (and some variants)

Handling Categorical Data

Converting to Numerical Values

- Data often contains non-numerical attributes. Machine Learning, however, requires numerical values in order to learn. Hence, must modify categorical attributes.

```
In [7]: housing["ocean_proximity"].value_counts()

Out[7]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

- Two categorical data types:
  - **Ordinal**: values can be sorted or ordered (e.g. shirt size: XL > L > M).
  - **Nominal**: text values without a order (e.g. shirt color: red, blue, black,…)

Handling Categorical Data

Converting to Numerical Values

- We can transform the values using Scikit-Learn’s OrdinalEncoder, which assigns a numeric value to each class

```
housing_cat = housing[['ocean_proximity']]
housing_cat.head(10)
```

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND
3555	<1H OCEAN
19480	INLAND
8879	<1H OCEAN
13685	INLAND
4937	<1H OCEAN
4861	<1H OCEAN

0

4

1

```
try:
    from sklearn.preprocessing import OrdinalEncoder
except ImportError:
    from future_encoders import OrdinalEncoder # Scikit-Learn < 0.20
```

```
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

```
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

Category	Value
<1H OCEAN	0
INLAND	1
ISLAND	2
NEAR BAY	3
NEAR OCEAN	4

Handling Categorical Data

Converting to Numerical Values

- We can transform the values using Scikit-Learn's OrdinalEncoder, which assigns a numeric value to each class

```
try:
    from sklearn.preprocessing import OrdinalEncoder
except ImportError:
    from future_encoders import OrdinalEncoder # Scikit-Learn < 0.20
```

```
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

```
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

Category	Value
<1H OCEAN	0
INLAND	1
ISLAND	2
NEAR BAY	3
NEAR OCEAN	4

```
housing_cat = housing[['ocean_proximity']]
housing_cat.head(10)
```

	ocean_proximity	
17606	<1H OCEAN	0
18632	<1H OCEAN	
14650	NEAR OCEAN	4
3230	INLAND	1
3555	<1H OCEAN	
19480	INLAND	
8879	<1H OCEAN	
13685	INLAND	
4937	<1H OCEAN	
4861	<1H OCEAN	

This gives Ordinal values, but ordering/similarities are not needed

Handling Categorical Data

One-hot Encoding

- To fix this, create one binary attribute per category (e.g. a binary vector), where only one non-zero value exists, based on the category

	ocean_proximity
17606	<1H OCEAN
18632	<1H OCEAN
14650	NEAR OCEAN
3230	INLAND
3555	<1H OCEAN
19480	INLAND
8879	<1H OCEAN
13685	INLAND
4937	<1H OCEAN
4861	<1H OCEAN

Category	Vector Value
<1H OCEAN	0
INLAND	0
ISLAND	0
NEAR BAY	0
NEAR OCEAN	1

Category	Vector Value
<1H OCEAN	1
INLAND	0
ISLAND	0
NEAR BAY	0
NEAR OCEAN	0

- This is called a ***one-hot encoding***, because only one value will be 1 (hot), which the others are 0 (cold).
- Avoids issues with ordering and similarity

Handling Categorical Data

One-hot Encoding

- One-hot encoding can be accomplished with Scikit-Learn using OneHotEncoder

```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

- Create instance of encoder
- Apply encoding to categorical data

```
cat_encoder.categories_
```

```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

- Shows what position in vector implies (e.g. which category)

Feature Scaling

Two approaches

- Machine learning algorithms do not perform well when the features/attributes have very different numerical scales
  - Total rooms varies from 2 to 39320
  - Median ages varies from 1 to 52
- ***<u>Feature scaling</u>***, modify the range of values while maintaining relative information, is needed. Two common approaches:
  - Min-max scaling (aka normalization)
  - Standardization

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000

Feature Scaling

Min-max scaling (or normalization)

- Min-max scaling (or normalization) involves:
  - Computing the min and max values of the attribute/feature
  - Subtract the min value from each instance of this attribute
  - Divide the result by the difference between the max and min values.
- Results in attributes/features that range from 0 to 1.

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
housing_rooms = housing[["total_rooms"]]
housing_rooms_scaled = scaler.fit_transform(housing_rooms)
print("Min: ", min(housing_rooms_scaled), "Max: ", max(housing_rooms_scaled))
```

```
Min:  [0.] Max:  [1.]
```

Feature Scaling

Standardization

- Steps for standardizing features:
  - Compute mean (or average) and standard deviation of feature/attribute
  - Subtract the mean value from each instance of this attribute
  - Divide the result by the standard deviation.
- Resulting attributes/features are zero mean and unit variance, but not bound to specific range.
- See StandardScaler in Scikit-Learn for a built-in function for accomplishing this.

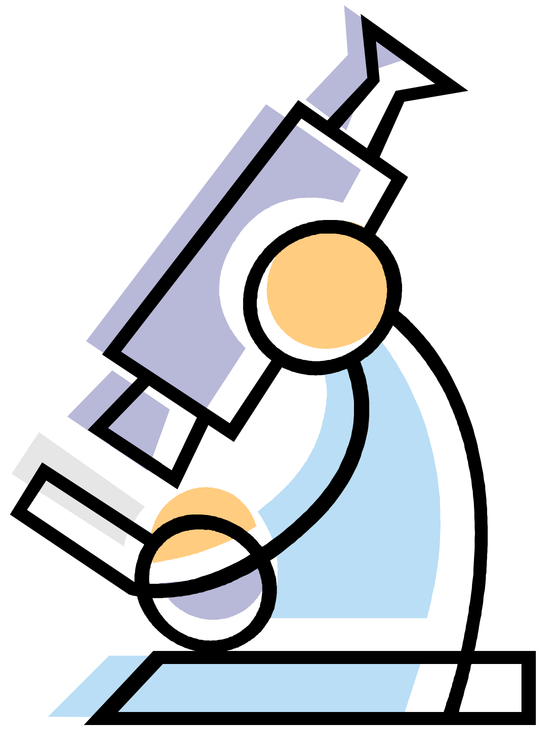
Evaluation and Cross Validation

Why Evaluation?

- When a learning system is deployed in the real world, we need to be able to quantify the performance of the system
  - **How accurate will the classifier be? How big is the regression error?**
  - **When is it wrong? Why is it wrong?**
- Evaluation is also needed during training/development for the same reasons
- This is very important as it is useful to decide which classifier/regressor to use in which situations

Evaluating ML Algorithms

Often done empirically (e.g. running experiments)



- Empirical Studies
  - **Correctness on novel examples**
  - Time spent learning
  - Time needed to apply result learned
  - Speedup after learning (explanation-based learning)
  - Space required
- <u>Basic idea</u>: repeatedly use <u>train/test</u> sets to estimate future performance

Proper Experimental Methodology Can Have a Huge Impact!

- A 2002 paper in Nature (a major, major journal) needed to be corrected due to “training on the testing set”

Most important “thou shall not”

- **Original report** : 95% accuracy (5% error rate)
- **Corrected report (which still is buggy)**: 73% accuracy (27% error rate)
- Error rate increased over 400%!!!

Recall: Training and Test Sets

Split data into two sets

- Split the available data into a training set and a test set

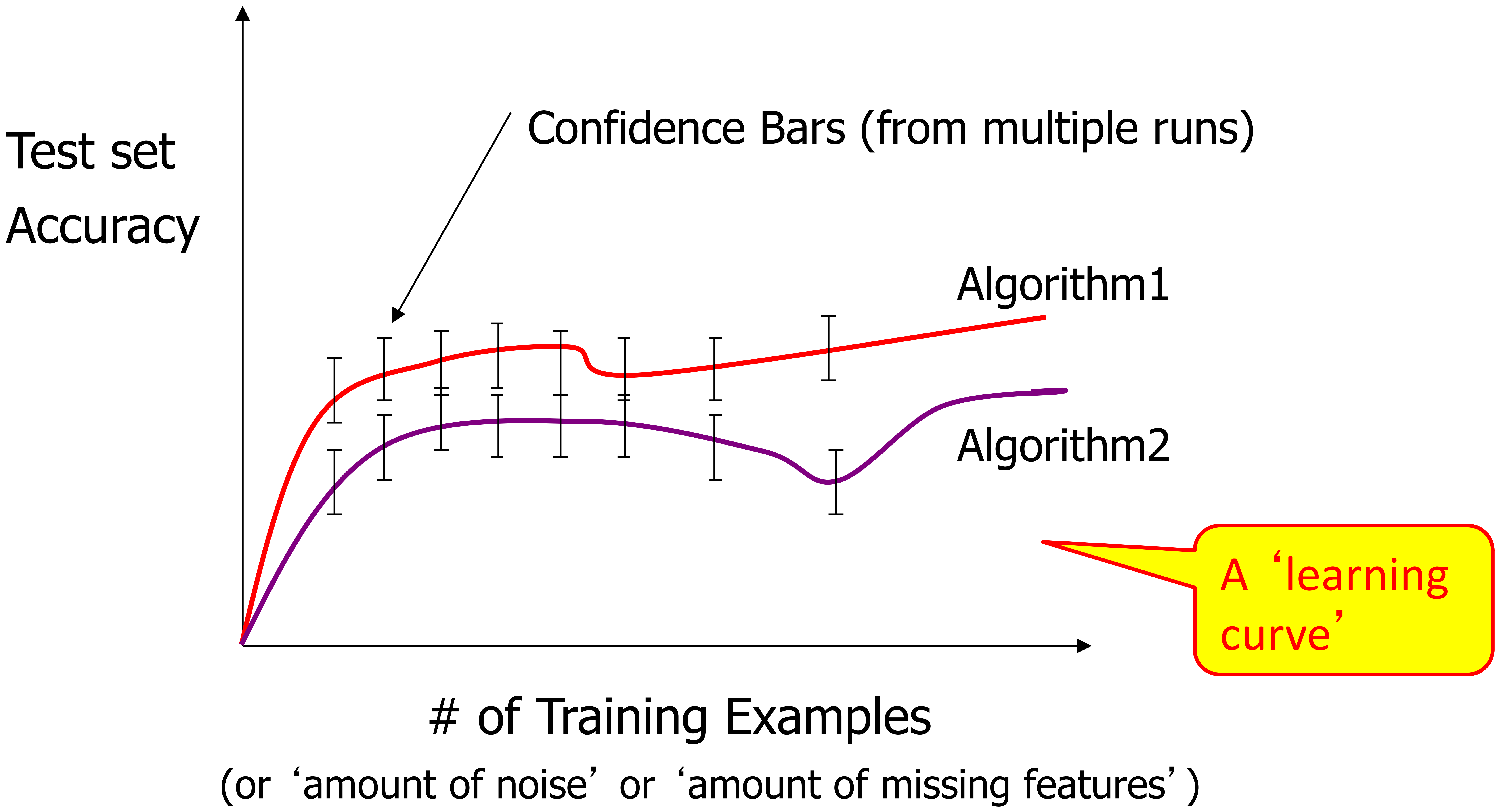
- Train the ML algorithm on the training set and evaluate it on the testing set
- Assume already performed data pre-processing
- Now want to train and evaluate a classifier (e.g. Linear Regression - Don’t worry about understanding what this is at this point)

Classifier Accuracy

- The accuracy of a classifier on a given test set is the percentage of test set examples that are correctly classified by the classifier
  - Accuracy = (# correct classifications)/ (Total # of examples)
  - Error rate is the opposite of accuracy
    - Error rate = 1 - Accuracy

Some Typical ML Experiements

Empirical Learning



Some Typical ML Experiments

“Lesion” Studies

	Testset Performance
Full System	80%
Without Module A	75%
Without Module B	62%
…	…

False Positive and False Negatives

- Sometimes accuracy is not sufficient
- If 98% of examples are negative (for a disease), the classifying everyone as negative can get an accuracy of 98%
- When is the model wrong?
  - **False positives** and **false negatives**
- Often there is a cost associated with false positives and false negatives
  - Diagnosis of diseases
  - Sometimes better safe than sorry

Confusion Matrix

- Is a device used to illustrate how a model is performing in terms of false positives and false negatives
- It gives us more information than a single accuracy figure
- It allows us to think about the cost of mistakes
- It can be extended to any number of classes

Confusion Matrix

- **True positive** is the count (or percentage) of instances where the model predicted class A, and class A is the true label (or result)

- **False Negative** is the count of instances where the model predicts class B, even though the true label is class A.

- **False Positive** is the count of instances where the model predicated class A, given that Class B is the true label

- **True Negative** is the count of instances where the model predicted class B given that class B is the true label.

Predicted result			
Class A	**Class B**		
True Positive (TP)	False Negative	**Class A (e.g. have disease)**	**True Result**
False Positive (FP)	True Negative (TN)	**Class B (e.g. do not have disease**	

Confusion Matrix

- Can obviously be extended to more than two-class problems. Think about how?
- Ideally, the highest counts are along the main diagonal

Predicted result			
Class A	Class B		
True Positive (TP)	False Negative	Class A (e.g. have disease)	True Result
False Positive (FP)	True Negative (TN)	Class B (e.g. do not have disease	

Accuracy Measures

Four common metrics for assessing classification performance

		Predicted result		
		Class A	Class B	
True Result	Class A (e.g.	True Positive	False Negative	
	Class B (e.g. do	False Positive	True Negative	

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

$$True\,Positive\,Rate\,(sensitivity) = \frac{TP}{TP + FN}$$

$$Misclassification\,Rate = \frac{FP + FN}{TP + FP + TN + FN}$$

$$True\,Negative\,Rate\,(specificity) = \frac{TN}{TN + FP}$$

Accuracy Measures

Two more measures

Predicted result			
Class A	Class B		
True Positive	False Negative	Class A (e.g.	True Result
False Positive	True Negative	Class B (e.g. do	

- **Precision** = (# of relevant items retrieved) / (total # of items retrieved)

  = TP / (TP + FP)

  $\cong$ P(is pos | called pos)

- **Recall** = (# of relevant items retrieved) / (# of relevant items that exist)

  = TP/(TP+FN) = TPR

  $\cong$ P(called pos | is pos)

Notice you get no credit for filtering out irrelevant items

Learning from Examples

Standard Methodology for Evaluation

- Start with a dataset of labeled examples
- Randomly (or Stratified) partition into N groups
- N times, combine N- 1 groups into a training set
- Provide training set to learning system
- Measure accuracy on "left out" group (the testing set)
- Repeat until all combinations are evaluated

Called N-fold cross validation (typically N =10)

N-fold Cross Validation in Python

```python
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = (y_train_5[train_index])
    X_test_fold = X_train[test_index]
    y_test_fold = (y_train_5[test_index])

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565 and 0.96495
```

N-fold Cross Validation in Python

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = (y_train_5[train_index])
    X_test_fold = X_train[test_index]
    y_test_fold = (y_train_5[test_index])

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565 and 0.96495
```

N=3 Stratified folds for cross validation

N-fold Cross Validation in Python

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = (y_train_5[train_index])
    X_test_fold = X_train[test_index]
    y_test_fold = (y_train_5[test_index])

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565 and 0.96495
```

N=3 Stratified folds for cross validation

Train and test over the different folds iteratively

N-fold Cross Validation in Python

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = (y_train_5[train_index])
    X_test_fold = X_train[test_index]
    y_test_fold = (y_train_5[test_index])

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565 and 0.96495
```

N=3 Stratified folds for cross validation

Train and test over the different folds iteratively

Compute accuracy

Using Tuning Sets

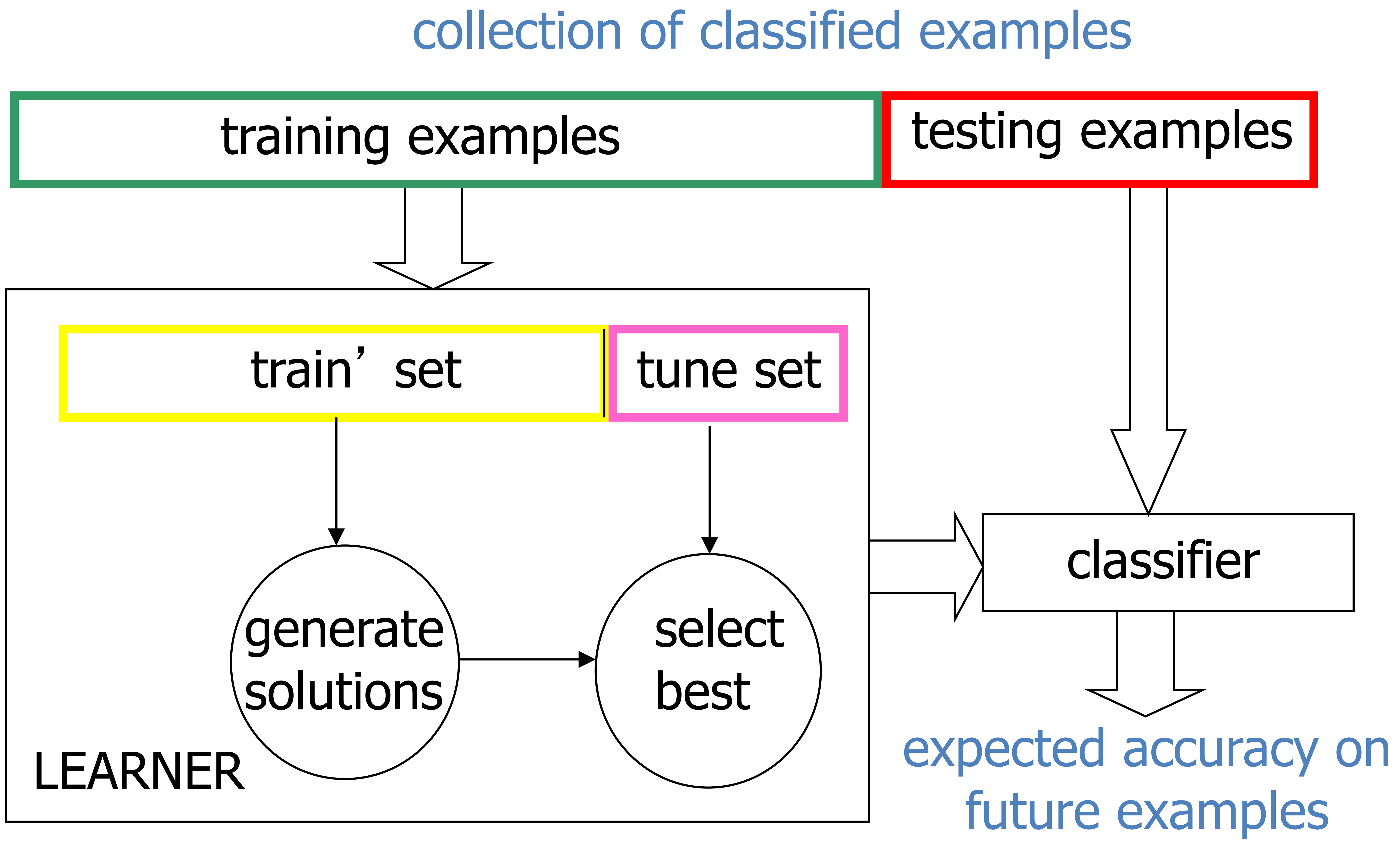
Refining N-fold Cross Validation

- Often, an ML system has to choose when to stop learning, select among alternative answers, etc.
- One wants the model that produces the highest accuracy on future examples ("overfitting avoidance")
- It is a "cheat" to look at the test set while still learning
- **Better method**
    - Set aside part of the training set, called a "tuning" or "development" set
    - Measure performance on this "tuning" data to estimate future performance for a given set of parameters
    - Use best parameter settings, train with all training data (except test set) to estimate future performance on new examples

Experimental Methodology

A Pictorial Overview

Statistical techniques such as 10-fold cross validation and *t*-tests are used to get meaningful results



Parameter Setting

- Notice that each train/test fold may get different parameter settings!
    - That’s fine (and proper)

- I.e., a “parameterless”* algorithm internally sets parameters for each data set it gets

- * Usually, through, some parameters have to be externally fixed (e.g. knowledge of the data, range of parameter settings to try, etc.)

Using Multiple Tuning Sets

- Using a single tuning set can be unreliable predictor, plus some data “wasted.”
- **Hence, often the following is done:**
  - For each possible set of parameters
    - Divide training data into train’ and tune sets, using N-fold cross validation
    - Score this set of parameter values: average tune set accuracy over the N folds
  - Use best set of parameter settings and all (train’ + tune) examples
  - Apply resulting model to test set

Next Class

Finish metrics for classification

Go over metrics for regression