DEPAUL UNIVERSITY

DSC478: Programming Machine Learning Applications

Roselyne Tchoua

rtchoua@depaul.edu

School of Computing, CDM, DePaul University

Week 6

GATHERING DATA

Learning Accuracy Depends on Data

Lots of potentially incorrect data, e.g., instrument faulty, human or computer error, transmission error

- incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
  - e.g., *Occupation*=“ ” (missing data)
- noisy: containing noise, errors, or outliers
  - e.g., *Salary*=“−10” (an error)
- inconsistent: containing discrepancies in codes or names, e.g.,
  - *Age*=“42”, *Birthday*=“03/07/2010”
  - Was rating “1, 2, 3”, now rating “A, B, C”
- Intentional (e.g., *disguised missing* data)
  - Jan. 1 as everyone’s birthday?

Learning Accuracy Depends on Data

- Lots of potentially incorrect data, e.g., instrument faulty, human or computer error, transmission error
  - Incomplete
  - Noisy
  - Inconsistent
  - Intentional
- Ignore bad data when possible, otherwise smooth-out, infer, correct …

Learning Accuracy Depends on Data

- Remember data pre-processing!
  - Normalized? Standardized? Easier to think about when it is age vs. income, but harder to think about when scales vary slightly between images of signals for example, but this may matter!
  - Convert categorical attributes to numerical features
  - Etc.
- Is the data representative of future novel cases – critical
  - Are the features relevant?
  - Proper representation?
  - Do you have domain knowledge?
- Visualization
- Do this before feature selection!
  - e.g., most selected method “Unknown” – This may be important to know but also maybe completely useless

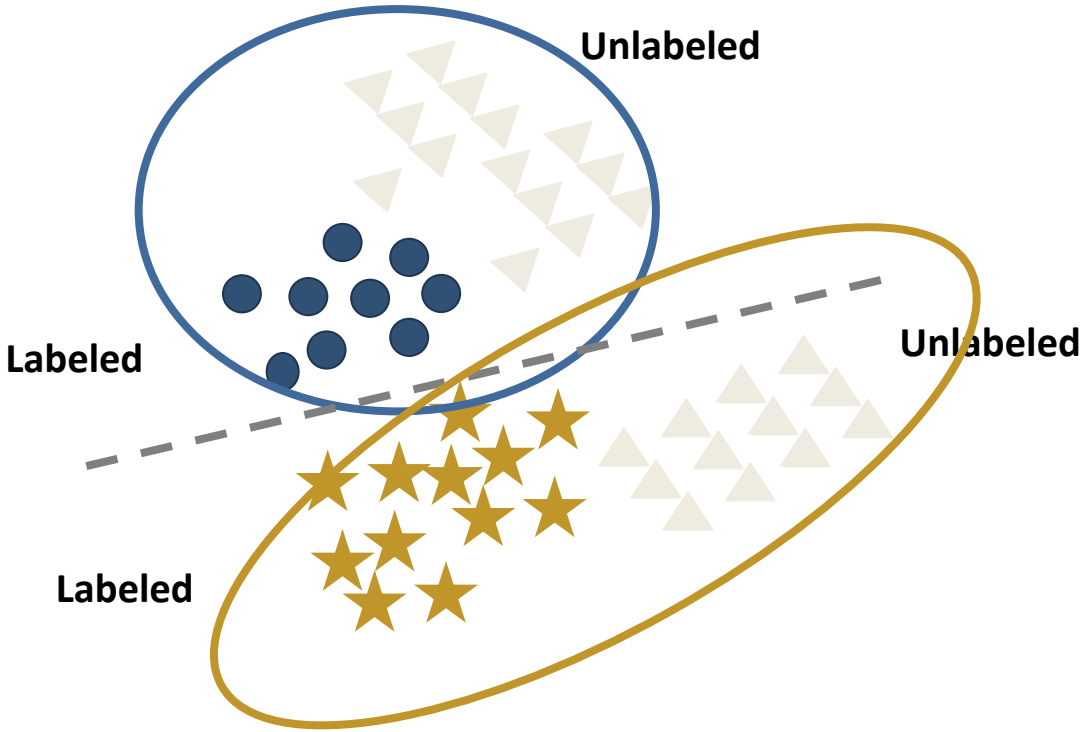
Gathering Data

- How much of the data is labeled (output target) vs. unlabeled – the more data, the better. Labeled data is best! Otherwise:
  - Set up (expert) labeling. Consider cost.
  - Crowdsourcing labeling is cheaper when possible – if task is easy and not proprietary
    - Amazon Turk: https://www.mturk.com/worker
    - Figure8: https://www.figure-eight.com/
  - Clustering
  - Active Learning
- Is the number of features/dimensions reasonable? The more features you have the more data you need!

Gathering Data: Semi-Supervised

1. Clustering to infer labels: Group data using categories in the initial data

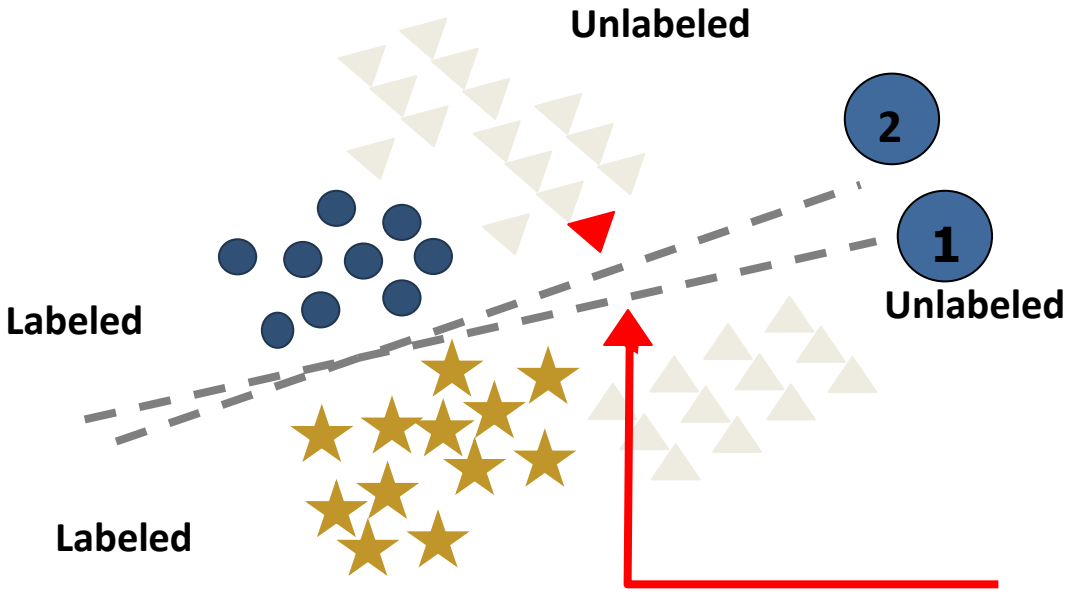
Interesting questions:
Could clustering reveal a third group/class?



Gathering Data: Semi-Supervised

2. Active Learning:
   - Let the computer ask for most informative labels (max uncertainty, close to decision boundary)
   - Ask an oracle = expensive experiment, paying database, expert, crowd etc.

- Build classifier with few labeled points available
- Classify all data, including unlabeled data
- Examine probabilities, i.e., uncertainty



When to Gather More Data?

When trying to improve performance, you may need

- More data
- More features
- Better input features (better representation)
- Better object/feature ration?
- Different machine learning models or hyperparameters
- etc.

When to Gather More Data?

- Hard question to answer. Need to factor in
  - Number of classes
  - Number of input parameters
  - Feature/Data ratio
    - About 25% of number of objects to be features, no more?
    - There must be x% more examples than there are input features, where x could be tens (e.g., 10).

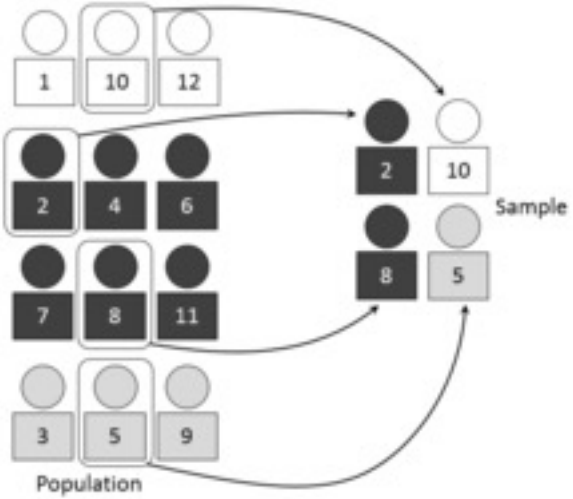
← Model Complexity Measures

- General guidelines
  - Compare your accuracy on training and test set
  - If bad training set accuracy then you probably need more and better data, features, noise, etc., or you might need a different learning model/hyperparameters
  - If test set accuracy much worse than training set accuracy then gathering more data is usually a good direction, though regularization or learning model/hyperparameters could still be a major issue. (SVM, Neural Net)

When to Gather More Data?

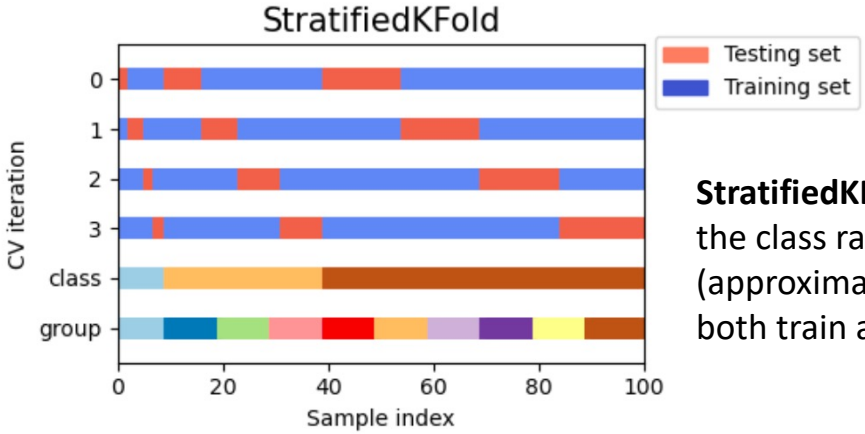
Imbalanced dataset

- Stratified Sampling
- Down or Under sampling (extreme cases)
- Over sampling (extreme cases) ***instead of gathering more***



https://en.wikipedia.org/wiki/Stratified_sampling

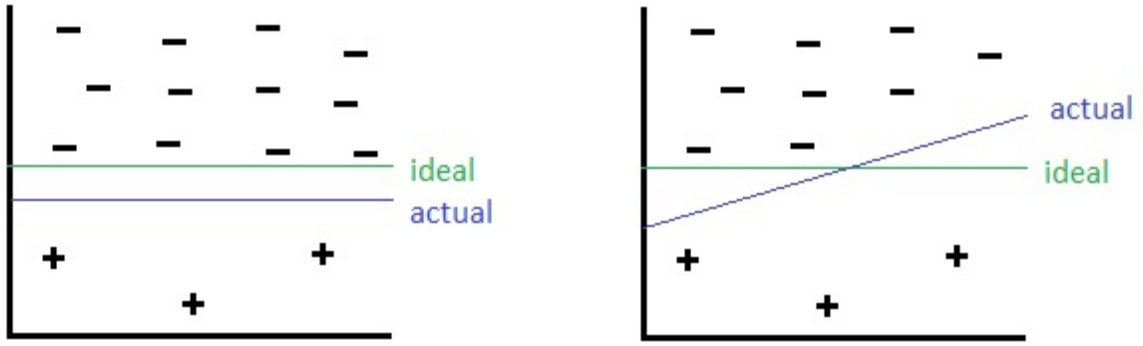
```
from sklearn.model_selection import StratifiedKFold
```



StratifiedKFold preserves the class ratios (approximately 1 / 10) in both train and test dataset.

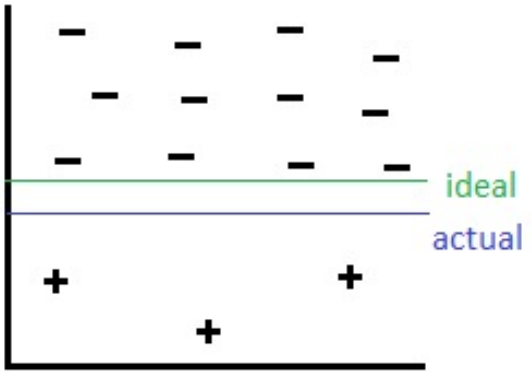
When to Gather More Data?

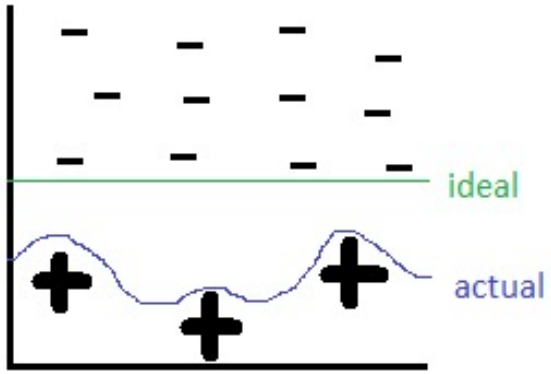
- Imbalanced dataset – Down/Under Sampling
  - Under sampling: randomly eliminate tuples from negative class.
  - Usually preferred, but there is a risk of removing corner negative cases or negative cases near the boundary



When to Gather More Data?

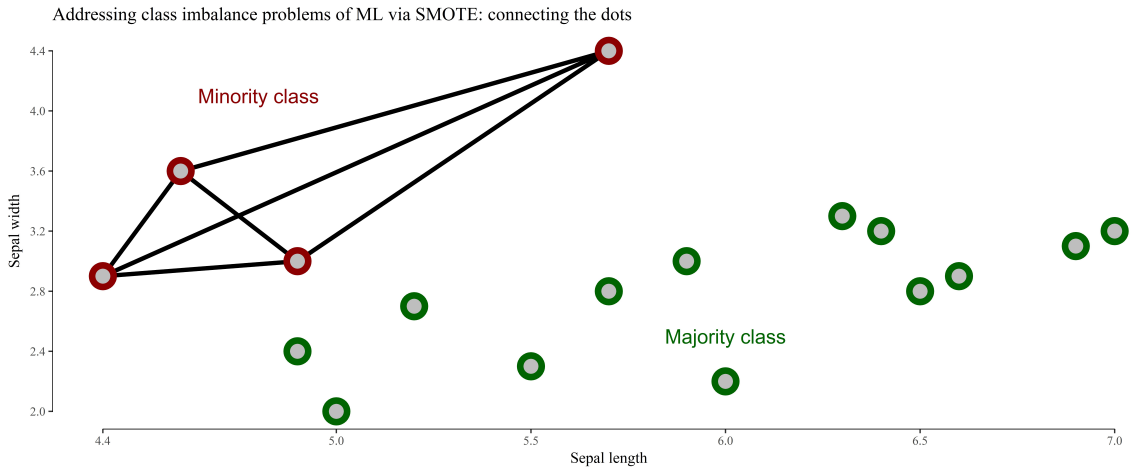
- Imbalanced dataset: Oversampling
  - re-sampling of data from positive class.
  - Just duplicating positive examples? Risk overfitting to training data





When to Gather More Data?

- Imbalanced dataset – Oversampling
  - Use something like Synthetic Minority Oversampling Technique (SMOTE) instead
    - **sudo pip install imbalanced-learn**
  - SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.



https://rikunert.com/SMOTE_explained

Week 6

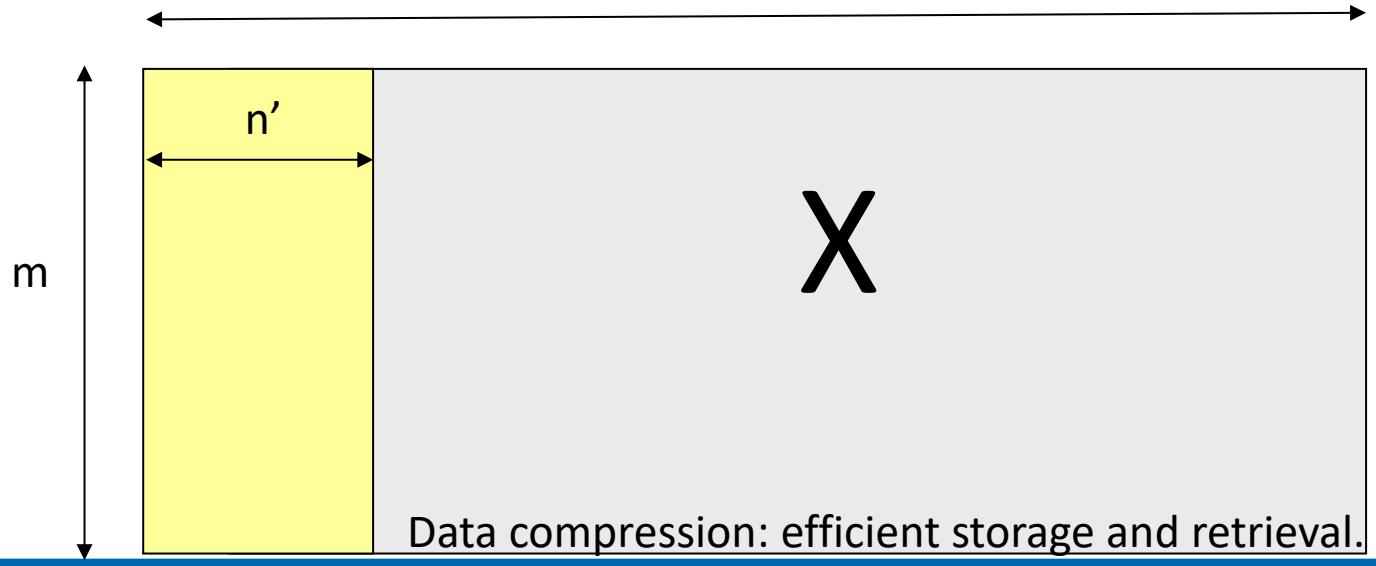
FEATURE AND PARAMETER SELECTION

FEATURE SELECTION

Why Feature Selection/Reduction?

Thousands to millions of low-level features: select the most relevant one to build better, faster, and easier to visualize/understand machine learning model

- The more feature, the more noise is introduced
  - The number of genes responsible for a certain type of disease may be small
- The more feature, the more data is required to learn



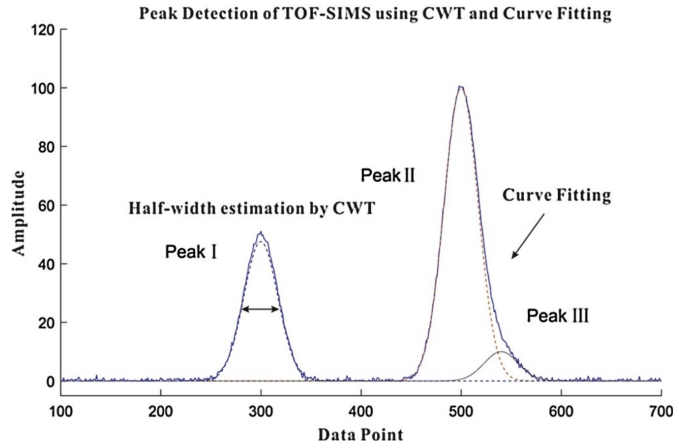
Need Better Features?

- Transforms of individual variables
  - Use area code rather than full phone number
  - Determine the vehicle make from a VIN (vehicle id no.)
- Combining/deriving variables
  - Height/weight ratio
  - Difference of two dates
- Features based on other instances in the set
  - This instance is in the **top quartile** of price/quality tradeoff
- This approach requires creativity and some knowledge of the task domain and can be *very* effective in improving accuracy!

Need Better Features?

This approach requires creativity and some knowledge of the task domain and can be *very* effective in improving accuracy!

- There is a sometimes a concern here of being too specialized and not generally useful, but it is extremely important to understand the question you're trying to solve.
  - Understanding the process is useful to other experts in the domain
  - Understanding will be useful to experts in similar fields



Peak Detection:
Work with 1400 points?
Work with features that make human sense?
Applicable to:
Materials Science
Medicine and more

(Ir)relevant Features

- Almost all instances have the same value (no information)
  - If there is a significant, though small, percentage of other values, then might still be useful
- Almost all instances have unique values (SSN, phone-numbers)
  - Might be able to use a variation of the feature (such as area code)
- Some features are highly correlated with other features
  - In this case the features may be redundant, and a subset are needed
- Careful if feature is too highly correlated with the target
  - Check this case as the feature may just be a synonym with the target and will thus lead to overfitting (e.g., the output target was bundled with another product, so they always occurred together)

Feature Selection and Reduction

Given n original features, it is often advantageous to reduce this to a smaller set of features for actual training

- Can improve/maintain accuracy if we can preserve the most relevant information while discarding the most irrelevant information
- and/or can make the learning process more computationally and algorithmically manageable by working with less features
- Curse of dimensionality requires an exponential increase in data set size in relation to the number of features to learn without overfit – thus decreasing features can be critical

Feature Selection and Reduction

- Feature Selection seeks a subset of the n original features which retains most of the relevant information
  - **Filters**, **Wrappers**
- Feature Reduction combines/fuses the n original features into a new smaller set of features which hopefully retains most of the relevant information from all features - Data fusion (e.g., LDA, PCA, etc.)

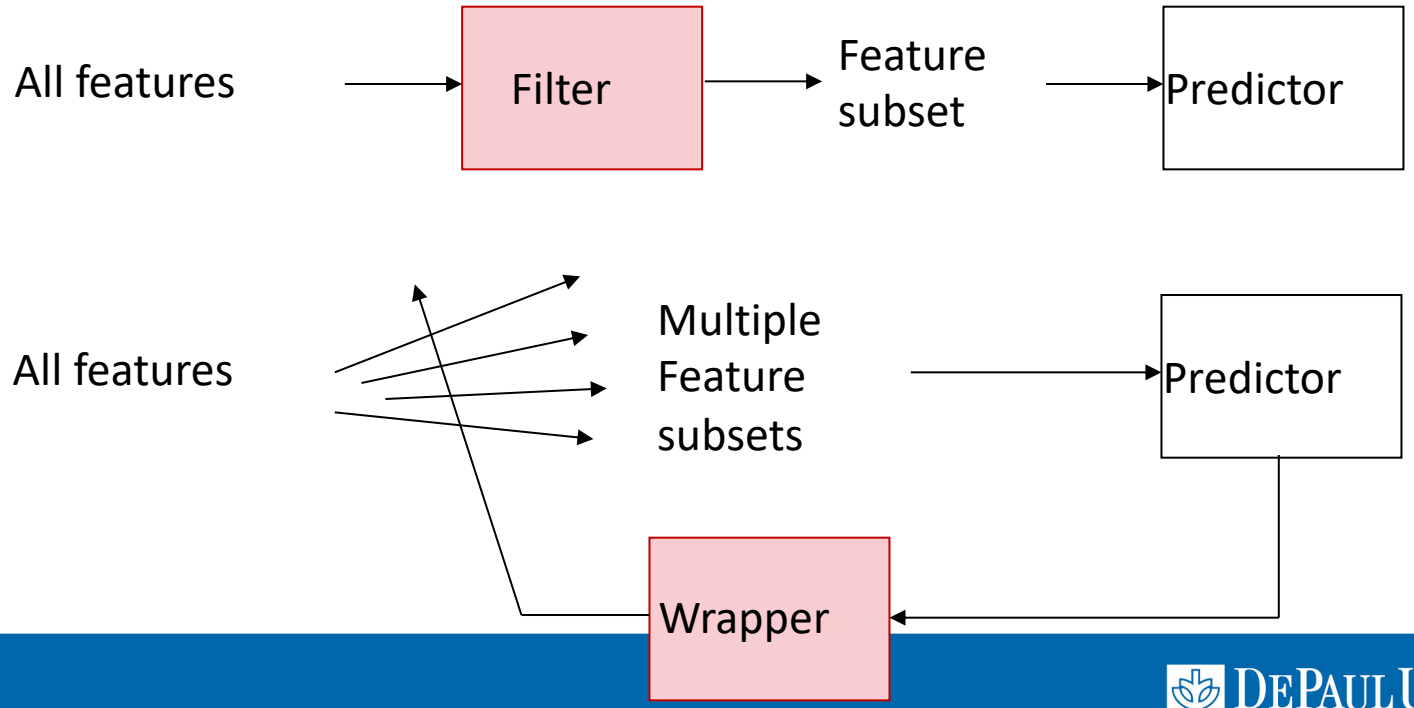
Feature Selection vs. Feature Reduction

- Feature Reduction
  - When classifying novel patterns, all features need to be computed, hard to tell which to drop apriori?
  - New features are combinations (linear for PCA/LDA) of the original features (difficult to interpret).
    - Example: term frequencies vs. Word2Vec numbers for feature vectors in text data
    - Example: X-ray intensity vs. PCA component – discuss problem with labeling.

- Feature Selection
  - When classifying novel patterns, only a small number of features need to be computed (i.e., faster classification).
  - New features is just a subset of the original features.

Feature Selection vs. Feature Reduction

- **Filter methods:** ranks features or feature subsets independently of the predictor (classifier), e.g., correlation coefficient with target
- **Wrapper methods:** uses a classifier to assess features or feature subsets.



Feature Selection - Filers

- Given n original features, how do you select size of subset
  - Attribute relevance analysis
  - User can preselect a size p (< n) – not usually as effective
  - Can find the smallest size where adding more features does not yield improvement
- Filters work *independently* of any particular learning algorithm
- Filters seek a subset of features which maximize some type of between class separability – or other merit score
- Can score each feature independently and keep best subset
  - e.g., 1st order correlation with output, fast, less optimal
- Can score subsets of features together
  - Exponential number of subsets requires a more efficient, sub-optimal search approach
  - Decision Tree or other ML model pre-process
  - How to score features is independent of the ML model to be trained on and is an important research area

Feature Selection - Wrappers

- Optimizes for a specific learning algorithm
- The feature subset selection algorithm is a "wrapper" around the learning algorithm
  - Pick a feature subset and pass it in to learning algorithm
  - Create training/test set based on the feature subset
  - Train the learning algorithm with the training set
  - Find accuracy (objective) with validation set
  - Repeat for all feature subsets and pick the feature subset which led to the highest predictive accuracy (or other objective)
- Basic approach is simple
- Variations are based on how to select the feature subsets, since there are an exponential number of subsets

Feature Selection - Wrappers

- Exhaustive Search – Exhausting, impractical
  - Heuristic – domain knowledge
  - Random search? Escape local optima

Cannot guarantee optimality

- Step-wise or Forward Search – Greedy
  - Score each feature by itself and add the best feature to the initially empty set FS (FS will be our final Feature Set)
  - Try each subset consisting of the current FS plus one remaining feature and add the best feature to FS
  - Continue until stop getting significant improvement (over a window)
- Step-wise Elimination or Backward Search – Greedy
  - Score the initial complete set FS (FS will be our final Feature Set)
  - Try each subset consisting of the current FS minus one feature in FS and drop the feature from FS causing least decrease in accuracy
  - Continue until begin to get significant decreases in accuracy

Feature Selection - Wrappers

- Exhaustive search e.g., exhaustive evaluation of 10 out of 20 features involves 184,756 feature subsets
- Forward and Backward Search O(n^2 · learning/testing time)
- Define Heuristics: define a "goodness" measure for feature, rank them, pick top d (doesn't consider combinations)
  - Basically, combine filter and wrapper methods
  - e.g., define uncertainty words, find pictures more likely to contain faces?
- Forward search could miss important higher order combinations. Backward search more robust to the higher order problem, since accuracy will drop if we drop an important feature, which may have not been selected in forward.
- Combining Forward and Backward Feature Selection
  - After adding some features, allow for some variables to be removed

Feature Selection – Filter vs. Wrapper

- Filters
  - Advantages:
    - Do not depend on the algorithm – solution may help families of classifiers
    - Fast execution – Does not require building a classifier
  - Disadvantages: Tends to select large subsets of data leaving the user to select an **arbitrary threshold**: variance, correlation coefficient..
- Wrappers
  - Advantages:
    - Usually achieve better accuracy since they are determined based on the algorithm
    - ***Avoids*** overfitting since they usually use cross-validation
  - Disadvantages:
    - Slow execution since you run the algorithm with each feature subset; can be come impractical
    - ***Lack of generality***! Not link to data but to algorithm, results may be completely different with a different algorithm.

Feature Reduction - PCA

- Principle Component Analysis (PCA) is one of the most common feature reduction techniques
- A linear method for dimensionality reduction
- Allows us to combine much of the information contained in *n* features into *p* features where $p < n$
- PCA is an ***unsupervised technique*** in that it does not consider the output class/value of an instance – There are other algorithms which do (e.g., Linear Discriminant Analysis (LDA))
- PCA works well in many cases where data has mostly linear correlations
  - The new dimension with the most variance is the first principal component
  - The next is the second principal component, etc.
- Meaning of feature may be lost or hard to interpret, different from combination

Feature Selection/Reduction Remarks

- Feature selection is and will remain an important issue in data mining, machine learning, and related disciplines
- Feature selection has a price in accuracy for efficiency
- Researchers need to have the bigger picture in mind, not just doing selection for the purpose of feature selection.
  - Materials science example

Feature Selection in Scikit-Learn

- RFE: Filter-based Backward elimination

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

- Classifiers like Decision Trees and Random Forest give Feature Importance

Feature Selection in Scikit-Learn

- RFE: Filter-based Backward elimination

The following example shows how to retrieve the 5 right informative features in the Friedman #1 dataset.

```
>>> from sklearn.datasets import make_friedman1
>>> from sklearn.feature_selection import RFE
>>> from sklearn.svm import SVR
>>> X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
>>> estimator = SVR(kernel="linear")
>>> selector = RFE(estimator, 5, step=1)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([ True,  True,  True,  True,  True, False, False, False, False,
       False])
>>> selector.ranking_
array([1, 1, 1, 1, 1, 6, 4, 3, 2, 5])
```

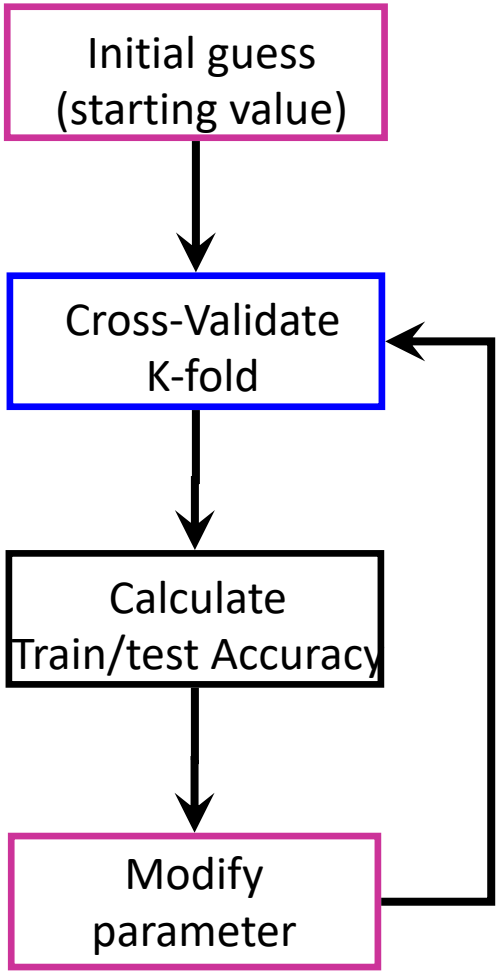
- Classifiers like Decision Trees and Random Forest give Feature Importance

Feature Selection Thoughts

- As the number of attributes increases, the likelihood of a learner to pick up on patterns that arise purely from chance increases
- In the extreme case where there are more attributes than data points (e.g., pixels in a video), even very simple hypothesis classes can overfit
  - Do humans just have great/better priors?
  - Are the best models, the ones that takes in directions/corrections from humans? Especially in scientific application!

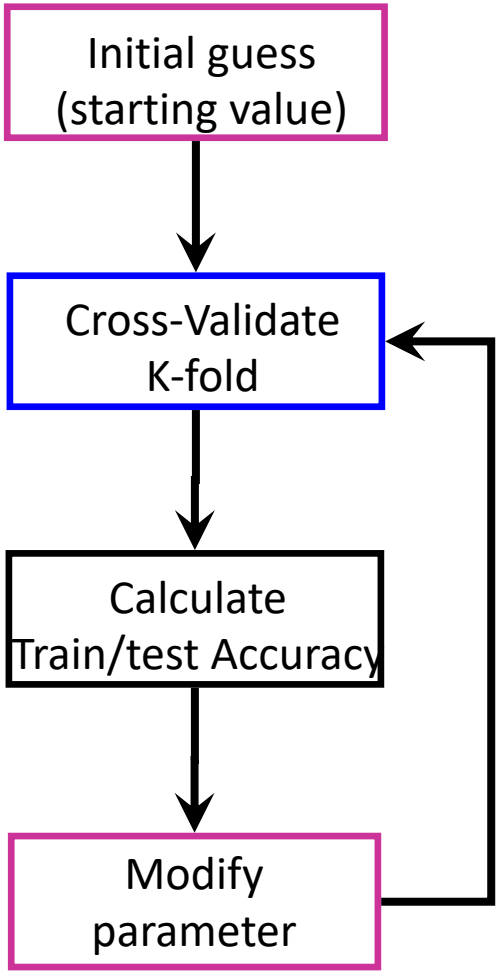
PARAMETER SELECTION

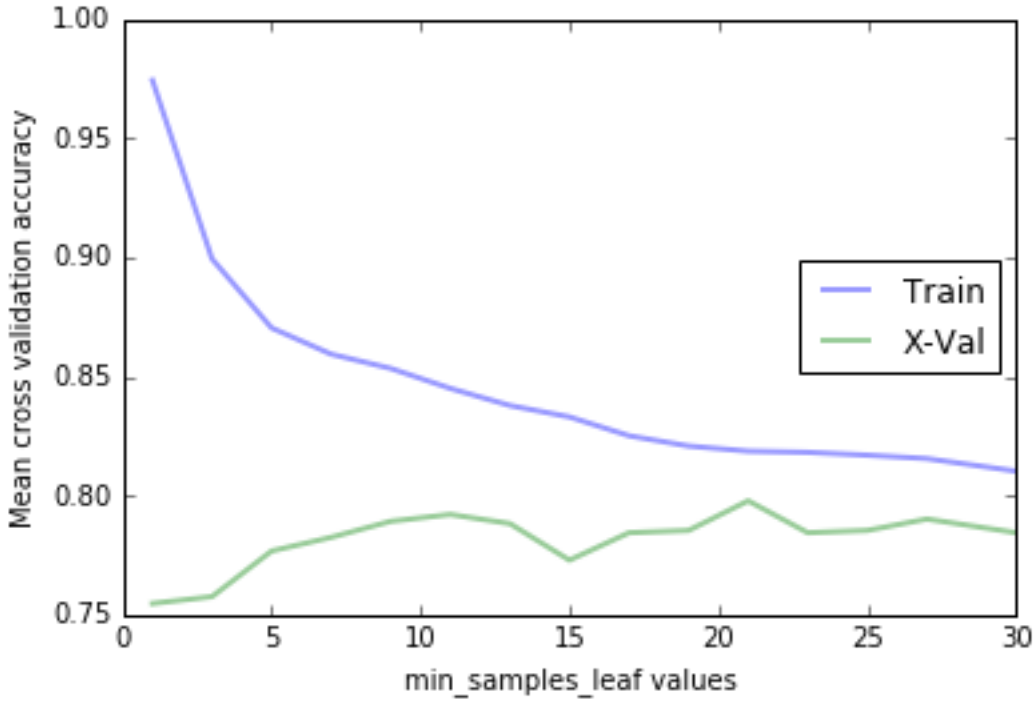
Parameter Selection via X validation



Parameter	TRAIN-ERR	10-fold-CV-ERR	Choice
K=1			
K=2			
K=3			
K=4			?
K=5			
K=6			

Parameter Selection via X validation





Parameter Selection via X Validation

- Generalize to multiple parameters with Grid Search
- Evaluate multiple parameters at the same time

```
depth_range = range(1, 10)
leaf_range = range(1,15)
param_grid = dict(max_depth=depth_range, min_samples_leaf=leaf_range)

d_tree = tree.DecisionTreeClassifier()
grid = GridSearchCV(d_tree, param_grid, cv=10, scoring='accuracy’)
grid.fit(X,y)print grid.best_score_

print grid.best_params_
print grid.best_estimator_
```

Parameter Selection

1. Split the training set into *K* smaller sets (under/over sample for unbalanced datasets)
2. Set aside each of the *K* folds one time for testing. Train as many models as there are different combinations of model parameters on the remaining *K-1* folds and compute the validation score on the hold-out fold.
3. For each set of parameters compute the mean validation score and select the parameter set with best performance on the hold-out validation sets.
4. Subsequently, train the model with the chosen parameter set on the full training set and estimate the generalization error using the evaluation set.

Parameters Sets with Continuous Values

- Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.
- The recipe below evaluates different alpha values for the Ridge Regression algorithm on the standard diabetes dataset. This is a one-dimensional grid search.

```
# Grid Search for Algorithm Tuning
import numpy as np
from sklearn import datasets
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
# load the diabetes datasets
dataset = datasets.load_diabetes()
# prepare a range of alpha values to test
alphas = np.array([1,0.1,0.01,0.001,0.0001,0])
# create and fit a ridge regression model, testing each alpha
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))
grid.fit(dataset.data, dataset.target)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```