Introduction to Supervised Machine Learning with Scikit-Learn

Greg Herman

AMS Short Course on Machine Learning in Python for Environmental Science Problems

Module II; AMS 99th Annual Meeting

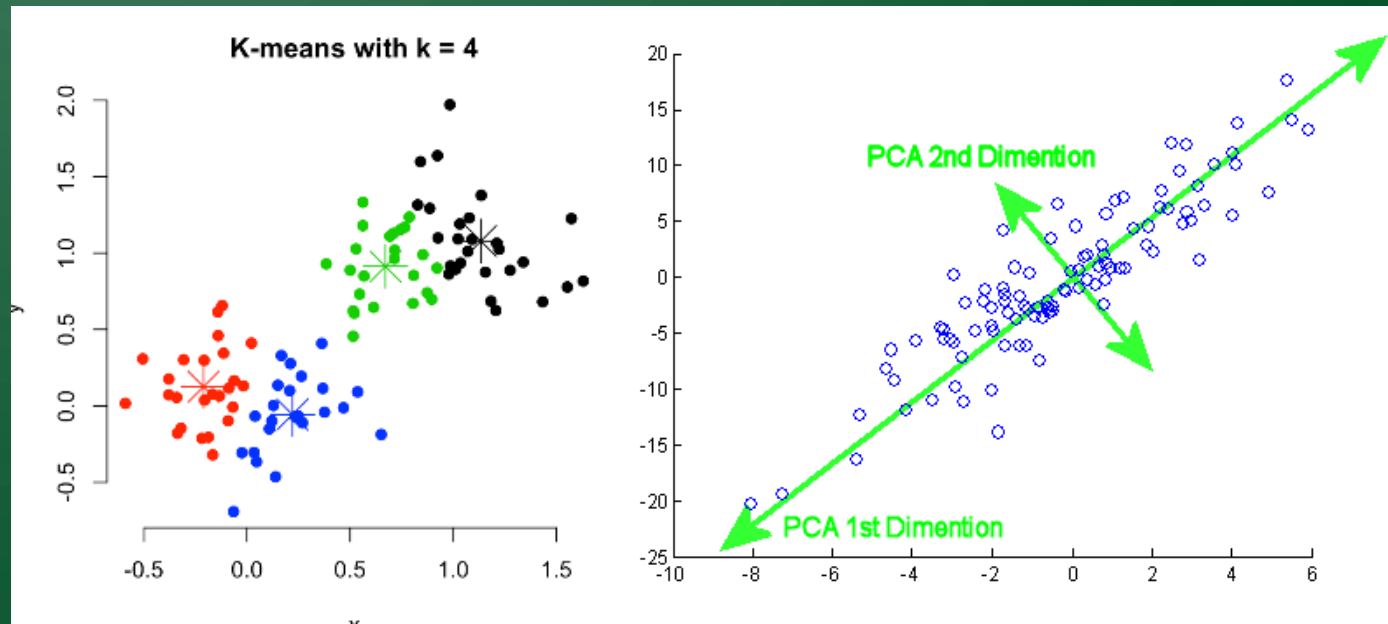
06 January, 2019

What Is Machine Learning? Why use it?

- Study of algorithms that learn structure from data in an automated fashion
- Can identify patterns and relationships in data that a human manually scrutinizing the data may miss
  - Machines can process data much more quickly than a human!
- Provides an automated, objective, and quantitative manner for diagnosing and correcting for model errors, biases, and other deficiencies
- Efficient manner to process all forecast guidance simultaneously in a streamlined manner
  - Impossible for operational forecasters to thoroughly inspect all of the guidance available to them in a real-time setting: too much to process, too little time!
- Has been successfully applied to a plethora of disciplines including computer vision, computational biology, natural language processing, finance, and medicine, among many others

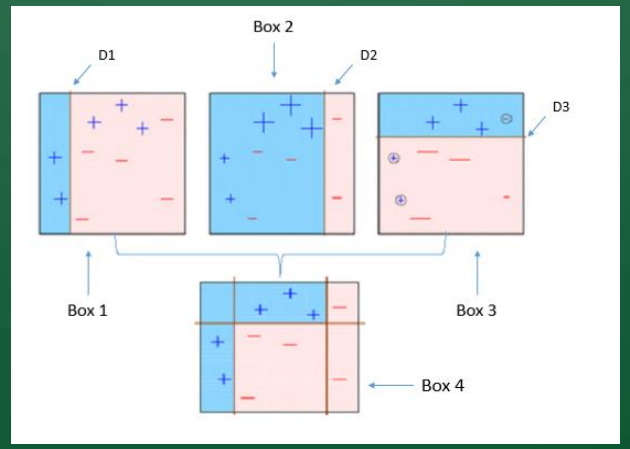
Supervised vs. Unsupervised Learning

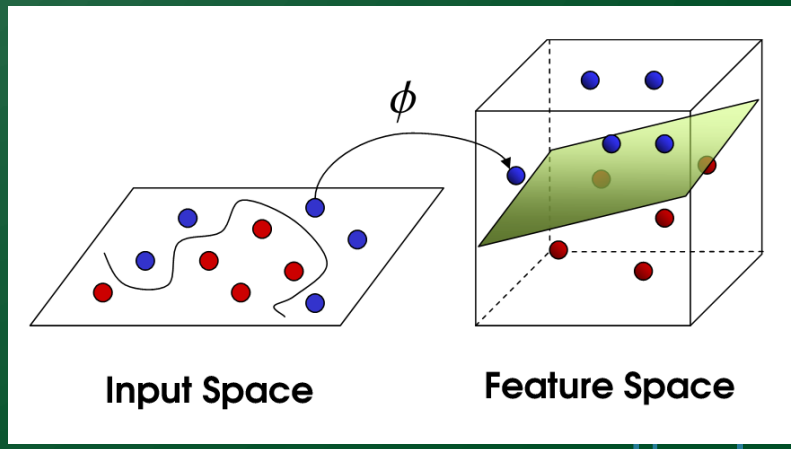
- Unsupervised Learning
  - Operates on *unlabeled* data
    - Either you're not trying to predict something explicitly, or you lack observations for what you're trying to predict
  - Deduces properties and/or commonalities from the examples themselves in order to restructure or reorganize
  - Examples:
    - Factorization Problems: PCA/EOF Analysis, CCA, ICA, NMF, etc.
    - Cluster Analysis
    - Density Estimation
    - Covariance Estimation
    - Neural Networks (sometimes)



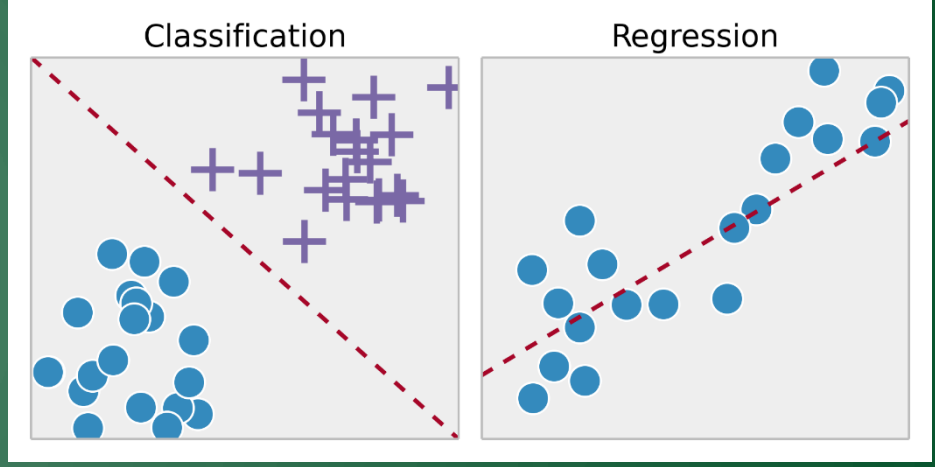
Supervised vs. Unsupervised Learning

- Supervised Learning
  - Operates on *labeled* data
    - Have many cases/records, all of which have both:
      - A set of variables that can be used to make an estimate/prediction
      - An observation of the quantity you wish to have a model estimate/predict (i.e. a label)
  - Learns and infers the structural relationship between your variables/predictors/features and the target/predictand of interest
  - Examples:
    - Regression
    - Classification





Regression and Classification



- Regression
  - Used for predicting *continuous* variables
  - Pros:
    - Output provides better context of projected magnitudes
    - Given an underlying regression problem, maintains resolution better than the analogous classification problem would afford
  - Cons:
    - Difficult to frame probabilistically despite inherent uncertainty to the problem
    - Forecasts may be subject to underperformance resulting from noisy data
    - Many atmospheric variables, especially those related to precipitation, exhibit mix of discrete and continuous distribution properties; it's difficult-to-impossible for regression algorithms to properly account for this
- Classification
  - Used for predicting *discrete* or *categorical* variables
  - Regression tasks can be readily reformulated as classification tasks
    - E.g. exceedance thresholds, physically or definitionally significant threshold
  - Pros:
    - More readily extends to probabilistic framework to allow for uncertainty quantification
    - Can be better tailored by model developer to account for critical thresholds of interest or concern
    - Sometimes more effective learning by grouping things known to be very similar together
    - Can better account for abnormal underlying predictand distributions
  - Cons:
    - Making regression tasks classification tasks can deteriorate learning if valuable intra-class signal is removed

Supervised Machine Learning Algorithms

- There are many choices!
- Each algorithm makes different assumptions about the relationship between individual predictors and those between predictors and the predictand.
- Each algorithm takes a different approach to learning structure from the supplied data

Steps to Conducting Machine Learning

1. Data acquisition/collection/generation
2. Data cleaning and pre-processing
3. Model training: model configuration, parameter tuning
4. Final model evaluation and interpretation

Data Cleaning and Pre-Processing

- Sheri has covered most of this
- A few things to keep in mind:
  1. Feature Imputation
     - In general, machine learning models cannot handle missing (NaN) values.
     - Model developer must handle missing values an appropriate way in pre-processing
       - Remove features with missing values
       - Remove training examples with missing feature values
       - Impute missing values to feature mean or some other *a priori* "best guess"
       - Impute missing values to some arbitrary small/large number
       - Impute missing values to zero
  2. Feature Standardization
     - Some algorithms are *scale invariant*, while many others are not.
       - Scale invariant algorithms care only about the qualitative relationships (e.g. greater than/less than); non-scale invariant algs depend also on the absolute *magnitude* of those differences
     - Predictive performance will typically suffer when using a scale-dependent model on non-standardized data
     - Standardization will often help facilitate learning, and rarely hurt
     - Before applying any algorithm to your dataset, think about how data should be pre-processed (standardized)
       - Not standardizing should always be a conscious choice!

Scikit-Learn

Scikit-Learn

- Open-source Python library for machine learning
- Overwhelmingly the most comprehensive ML Python library
- Heavily used across all fields and sectors
  - Describing paper published 2011; cited ~13,500 times!
- Generally quite well documented
  - User Guide: https://scikit-learn.org/stable/user_guide.html
    - Gives some mathematical/scientific background for how algorithms operate, effects of parameter options
  - API: https://scikit-learn.org/stable/modules/classes.html
    - Technical descriptions of all models/classes, their functions, and descriptions of arguments that can be supplied

Supervised Learning in Scikit-Learn

- All models share a common set of functions, making it very easy to try different models on your dataset(s)
- Here are the three-four main functions:

1. fit(X,y)
   - X: a 2-D numpy array (num_training_examples x num_features)
   - y: 1-D numpy array (num_training_examples) of target/predictand values associated with each example in X
   - This function does all the heavy lifting of training your ML model in one line!
   - Once complete, your model will be able to make predictions of the target based on unseen feature values.
   - This is usually the most computationally-intensive step of the process.

2. predict(X)

- X: a 2-D numpy array (num_validation_examples x num_features)
- This function takes a set of new examples and, using the fitted model, provides deterministic predictions of the target for each example

2b. predict_proba(X)

- This function exists only in Classification models
- Input is the same as predict()
- Output is 2-D numpy array (num_examples x num_classes), and gives for each example probabilities that the verification value lies in each event class.
- In general, output of predict() is the index of the maximum value for each row of the output of predict_proba()

3. score(X, y, [sample_weight])

- Evaluates R^2 (regression) or Accuracy Score (classification)
- I generally recommend applying your own evaluation function; at a minimum, think carefully about what you are trying to optimize before you validate using that metric.
  - R^2 is occasionally a metric of interest; Accuracy Score is almost never an appropriate way to validate a classification model in geoscientific applications.

Saving/Loading Models

- If you spend a long time training a model, presumably you want to use it again!
- In scikit-learn, this is most commonly done with joblib
- Importing: from sklearn.externals import joblib
- Saving: joblib.dump(mdl, output_path)
- Loading: joblib.load(output_path)
- WORD OF WARNING: scikit-learn is a fantastic machine learning library, it is still under active development and is constantly changing. Tread very carefully when switching between versions of sklearn. Saved models trained using an old version of sklearn will *not* in general be able to be loaded using a newer version, and even if they do, there is no guarantee that the functionality will necessarily be the same.

Supervised Learning Algorithms

Linear Regression

- Idea: Want to know some variable y
- Instances: m records of form $< y, [1, x_1, \ldots, x_n] >$
- m = number of records
- n = number of predictors/features
- Combine to form:
- $\boldsymbol{y_{u_{ijz}}} = \boldsymbol{\beta X} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\beta}$ is a weights vector of length n, and ε is the error term vector of length m

Please note: vector notation throughout largely dropped for simplicity. Unless otherwise noted, all vector products denote the *inner* product

Linear Regression: Computing β

- Many possible methods to estimate optimal coefficients for $\boldsymbol{\beta}$
- Simplest and most popular method is Ordinary Least Squares (OLS)
- Idea: Minimize total (summed) squared error (residuals) of predictive linear model:
- SSR(β) $= \sum_{q=1}^{M}(y_q - \boldsymbol{x_q\beta})^2 = \sum_{q=1}^{M} {\varepsilon_q}^2$
- $\widehat{\boldsymbol{\beta}} = argmin\big(SSR(\boldsymbol{\beta})\big)$

To find $\widehat{\boldsymbol{\beta}}$, differentiate $SSR(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ and set to zero.

$$0 = \frac{d}{d\boldsymbol{\beta}}\sum_{q=1}^{M}(y_q - \boldsymbol{x_q\beta})^2 = \frac{d}{d\boldsymbol{\beta}}\sum_{q=1}^{M} {y_q}^2 - 2y_q\boldsymbol{x_q\beta} + \boldsymbol{{x_q}^2\beta}^2 = \sum_{q=1}^{M}\frac{d}{d\boldsymbol{\beta}}({y_q}^2 - 2y_q\boldsymbol{x_q\beta} + \boldsymbol{{x_q}^2\beta}^2)$$
$$= 2\big(\boldsymbol{{x_q}^2\widehat{\beta}} - y_q\boldsymbol{x_q}\big)$$

$\boldsymbol{x^2\widehat{\beta}} = \boldsymbol{yx}$ (note: x is MxN, y is 1xM)

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{x^T x})^{-1}\boldsymbol{x^T y}$$

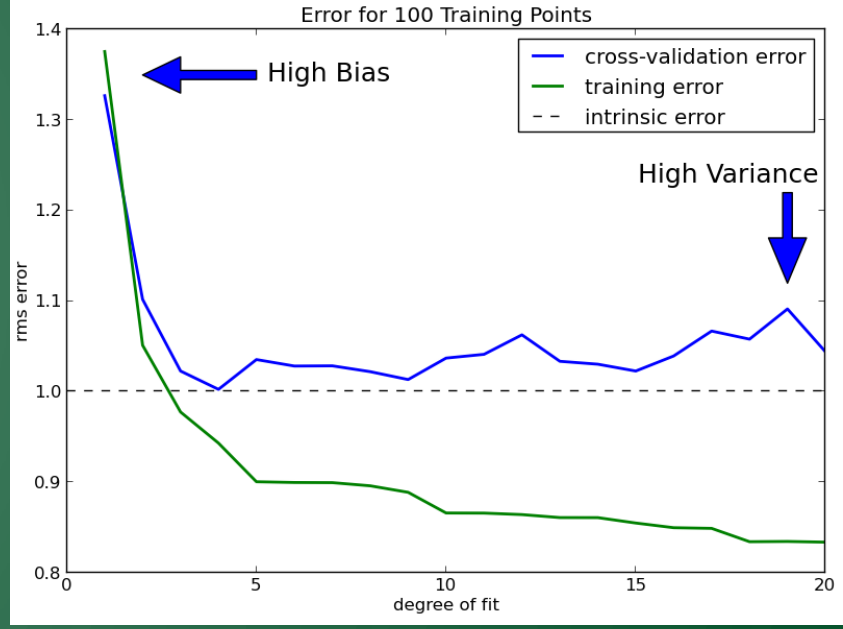
Linear Regression: Assumptions

- Predictand is a linear combination of the predictors (i.e. the predictand/predictor relationship is both linear and additive)
  - Not as significant of an assumption as one may initially think; variables may transformed to preserve a linear relationship.
    - Example: Temperature T is measured (or predicted in a model) at some location. That one 'observation' may serve as many predictors in the linear model:
      - $x_1 = T$
      - $x_2 = T^2$
      - $x_3 = T^4$
      - $x_4 = \ln T$
      - $\ldots$

Linear Regression: Assumptions

- Exogeneity: $E[\varepsilon|\boldsymbol{X}] = 0$
  - Residuals are uncorrelated with the predictors
  - Mean (expected) error is zero
- No perfect multicollinearity
  - Recall in my derivation of the OLS estimator for $\widehat{\boldsymbol{\beta}}$, matrix inversion was required to recover the estimate for $\widehat{\boldsymbol{\beta}}$
  - Recall from linear algebra that the following statements are equivalent for an NxN matrix A:
    - Matrix A is invertible
    - Matrix A's determinant is non-zero
    - Matrix A is non-singular
    - Matrix A has rank N
    - The columns of A are linearly independent
  - Long story short: If two predictors are perfectly correlated, then two columns of the predictor matrix will be linearly dependent and the predictor matrix will be singular and thus non-invertible
- Homoscedasticity
  - Variance in the residuals (errors) is *independent* of:
    - Time
    - Predictand Value
    - Any Predictor Value
- Independence of Residuals
  - Predictand is uncorrelated with residuals
  - Weaker than 'pure' statistical independence
- Residuals are normally distributed

Bias/Variance Tradeoff



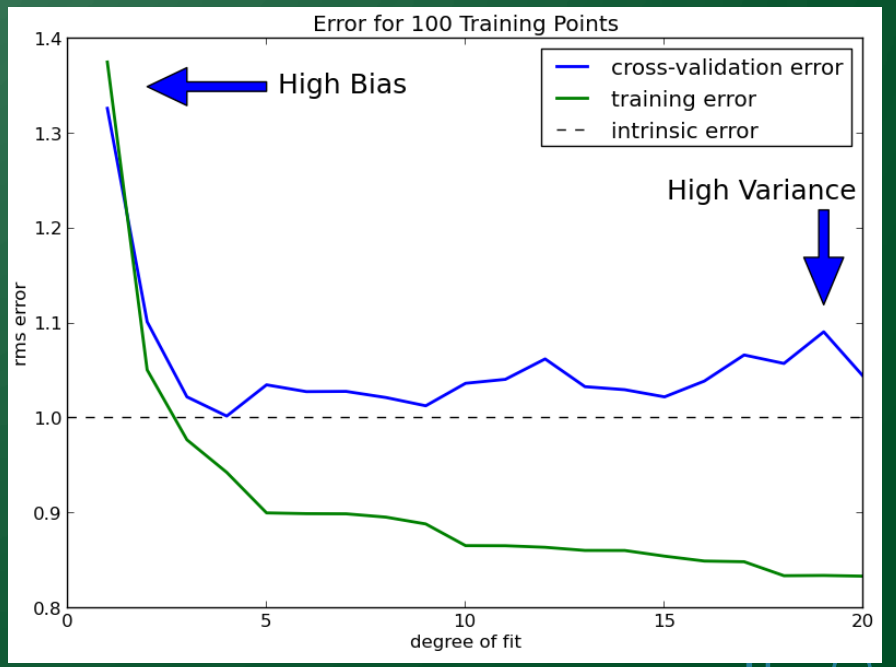
- Critical concept in statistics and machine learning
- Prediction error always arises from one of three causes:
  1. Bias: error that results from erroneous assumptions made by the learning algorithm
     - e.g. linear regression owhen the true predictor-predictand relationship is quadratic; not including a helpful predictor in the model, not incorporating codependent relationships, etc.
  2. Variance: error that results from sensitivity to changes in the training data
     - Results from *overfitting*: fitting the noise of the training data in addition to the signal
  3. 'Irreducible': error that results from noise in the fundamental underlying relationship between observables and the predictand
- Ultimately want to minimize total error. Can't do anything about irreducible error, so try to minimize sum of error resulting from bias and variance
- However, there is a tradeoff between bias and variance; increasing model complexity will ease assumptions made by the model and thereby reduce model bias, but will also increase model variance

Overfitting

- Ultimate Goal: Minimize Test Error $error_{true}$, however that is quantified
- Mathematically, a model is overfit if it outputs a solution $\widehat{\boldsymbol{\beta}}$ when ∃ a solution $\boldsymbol{\beta}^*$ such that:

$$\left(error_{train}(\widehat{\boldsymbol{\beta}}) < error_{train}(\boldsymbol{\beta}^*)\right) \wedge$$
$$\left(error_{true}(\widehat{\boldsymbol{\beta}}) > error_{true}(\boldsymbol{\beta}^*)\right)$$

- Training Error is *not* a good proxy for Test/True Error



Combatting Overfitting

- Approaches to avoid overfitting:

1. Start with a simple model—one that imposes a simple structure on the relationships between the predictors/predictand
2. Explicitly penalize model complexity
3. As a pre-processing step, reduce the dimensionality of the input data through *feature selection* or *feature extraction*
4. Use a lot of training data

Dimensionality Reduction

- In some situations, you may have a large number of candidate features, many of which are strongly correlated (by physical or spatiotemporal relationships, etc.)
- In these situations, it is easy for a learning model to fit to the statistical noise that produces differences between two correlated features rather than the underlying physical relationships
- A large feature set can also cause computational issues, making model training slow or impossible
- Two classes of approaches to alleviate these issues are *feature selection* and *feature extraction*

Feature Selection

- In feature selection, one selects some subset of the candidate features to use for the learning model
- There are three broad classes of feature selection approaches:
  1. Filter Methods
     - Filter methods use some metric to score each feature, and then either retain the top N features according to their scores, or retain all features exceeding a certain score
     - Candidate features are often treated independently in these methods, and sometimes are treated independent of the target as well
     - These are generally much more efficient than Wrapper Methods, though often not as effective
     - Some examples are variance thresholds, chi-squared selection, and mutual information selection
  2. Wrapper Methods
     - In these methods, the optimal feature set is treated as a search problem; different subsets of features are taken, a predictive model is trained, and evaluated, and then the feature subset is perturbed (usually either by adding or removing a feature). The model is retrained and re-evaluated successively until an 'optimal' subset is obtained
     - Since there are $2^{N\_features}$ possible feature sets, an exhaustive search is seldom if ever practical, and methods are generally implemented to evaluate only a small subset of possible feature sets
     - Some examples are sequential forward selection, sequential backward selection, or recursive feature elimination
  3. Embedded Methods
     - These methods are directly integrated into the model training phase, and the feature selection occurs in tandem with training
     - One primary example of this is Regularization

Feature Selection: Wrapper Methods

- There are many wrapper methods; I'll briefly discuss three of them:

1. Sequential Forward Selection (SFS)
   - Define some model performance measure (e.g. RMSE, Brier Score)
   - Start with an empty set of retained features, try adding each candidate feature separately, training and then evaluating (using the above-defined performance measure) each possible single feature model. Select the feature corresponding to the best performing single feature model, add to final model's feature set
   - Repeat the process assessing the additional utility supplied by adding a single remaining candidate feature to the set, adding the identified most useful feature to the final feature set each iteration
   - Repeat until the addition of any remaining candidate feature fails to improve predictive performance by some user-specified amount (can be zero)
   - This is *not* implemented in scikit-learn (yet)
2. Sequential Backward Selection (SBS)
   - This operates like SFS in reverse
   - Start with retaining every feature; try eliminating each feature and evaluate each all-but-one model
   - Select the feature set which gives the best predictive performance and continue to the next iteration
   - This is *not* implemented in scikit-learn (yet)
3. Recursive Feature Elimination (RFE)
   - Operates just like SBS, except instead of using a performance metric, importance measures like coefficient magnitude (for regression models) or feature importances (for tree-based models) are used instead to determine the least useful feature.
   - This *is* implemented in scikit-learn in the sklearn.feature_selection module

Feature Extraction

- In contrast to Selection, Feature Extraction seeks to redefine a new, smaller set of latent features, each represented as a function of existing features
- There are many methods available to perform feature extraction. A non-exhaustive list:

1. Principal Component Analysis (and derivatives)
2. Independent Component Analysis
3. Partial Least Squares Regression
4. Linear Discriminant Analysis

Some of these (e.g. PLSR, LDA) are predictand-specific, others (e.g. PCA, ICA) are not

These four are all implemented in sklearn.decomposition, sklearn.cross_decomposition, or sklearn.discriminant_analysis

Regularization

- In regression models, *regularization* is a process used to combat overfitting and improve the generalizability of the trained model to unseen examples
- In particular, this is accomplished by adding an explicit penalty model complexity (large weights, in the case of regression models) in the optimization problem
- Recall that regular ordinary least squares linear regression finds the weight or coefficient vector w that satisfies: $\min_w ||Xw - y||_2^2$
- Regularization adds an extra term (or more) to the minimization problem, the specifics of which depend on the kind of regularization employed:
  - Ridge
    - Uses L2 (Euclidean) Regularization
    - Penalizes large weights

    $\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$

    $\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$
  - LASSO
    - Uses L1 (Manhattan) Regularization)
    - Penalizes non-zero weights

    $\|\boldsymbol{x}\|_1 := \sum_{i=1}^{n} |x_i|.$

    $\min_w \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha||w||_1$
  - Elastic Net
    - Uses combination of L1 and L2 regularization

    $\min_w \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$

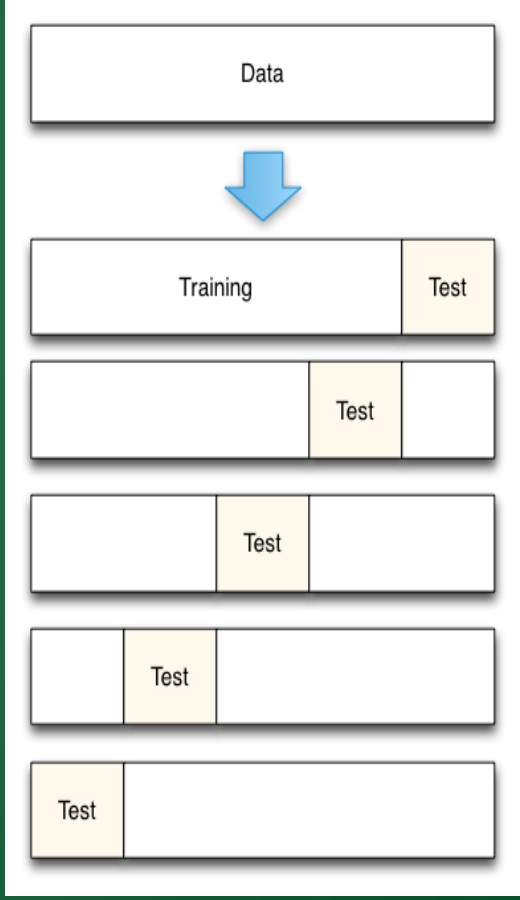
Regularized Linear Regression: Parameters

- Regularization introduces tunable parameters that affect kind and strength of regularization
- alpha: higher value -> more regularized (0 = no penalty)
- l1_ratio: Balance between L1 (LASSO) and L2 (Ridge) penalization (0 = Ridge; 1 = LASSO)

Ridge	LASSO	ElasticNet
alpha	alpha	alpha
		l1_ratio (rho)

- How to go about tuning parameters? What are the optimal values?

Validation



- In Data Pre-Processing, must perform Data Segmentation
- Two approaches:
  1. "Traditional Validation": Partition data into 3 sets: Training, Validation, Test
     - Always train your model on the Training set, and evaluate the performance on the Validation set. Tune configuration and parameters to optimize performance on the Validation Set
  2. "Cross Validation": Partition data into 2 sets: Training, Test
     - Further break up Training set into k chunks (K-fold Cross Validation)
     - For a given configuration/parameter settings, train k different models, each identical each withholding one of the k chunks from the training data to retain for validation
     - By evaluating all k models on their respective withheld chunks, Training set serves as both a Training and Validation set
- Cross-Validation makes more efficient use of dataset; more computationally expensive
- ***NEVER*** evaluate your model on its own training data!

Validation

- What do you actually do in the training/validation cycle?
- Feature Selection
- Parameter Tuning
  - All models have a set of tunable parameters
  - Number and specifics depend on the algorithm
    - Some algorithms are more sensitive to parameter values than others; some parameters have more effect than others
  - Most make a tradeoff between model *bias* and model *variance* as the parameter value is varied
  - Key to parameter tuning is to find “sweet spot” of parameter values where skill is maximized

Parameter Tuning

- How do you find the best parameter settings?
- No universal 'best' values–they will be application-specific
- Most common approach is a "Grid Search" approach
  - For the N parameters being tuned, devise an approximately evenly-spaced (linear or logarithmic, depending on the parameter) set of values spanning the set of plausible values for each parameter
  - Combine each of these N scales to form an N-dimensional grid of all reasonable combinations of model parameters
  - Evaluate performance of each set of parameter settings; select the best-performing setting
- Scikit-learn has built in capability to perform Grid Search using Cross-Validation
  - This is what ModelNameCV objects in scikit do
  - Not all methods have such an implementation!
- This is "best" approach, but computationally expensive and often impractical
- Alternative approach is "Greedy Method"
  - Start with one reasonable setting, and evaluate.
  - Perturb one parameter across set of reasonable values while holding the others constant; evaluate each setting
  - Set first parameter to value with most skillful verifying setting
  - Move on to second parameter; test over range of plausible values while holding first and all other parameters constant
  - Repeat until all parameters perturbed and optimal settings chosen
  - Cycle back through parameters; perturb values in the vicinity of values selected in first pass, reevaluate skill and update parameter values as appropriate
  - Repeat until further perturbations have little or negative impact on model performance

Test Data

- Test data should remain completely isolated and untouched for all model development
- Once model configuration/parameter settings finalized, retrain model on the entire Training+Validation set with those same settings
- Apply same evaluation functions used during validation to retrained model using the withheld Test Data
- These are the final performance metrics for your model

Classification

Logistic Regression

Logistic Regression

- Name is a misnomer; it is a Classification algorithm, not Regression algorithm
- Nevertheless, they share much in common
  - Both are instances of the Generalized Linear Model (GLM)
  - Three components to a GLM (Skipping over many of the details)
  1. A linear predictor η
  2. Probability Distribution of the Predictand Mean
  3. Function g Linking the Predictor (η) and the Mean of the Distribution Function (μ)
- Differences in underlying assumptions/formulation

Logistic Regression

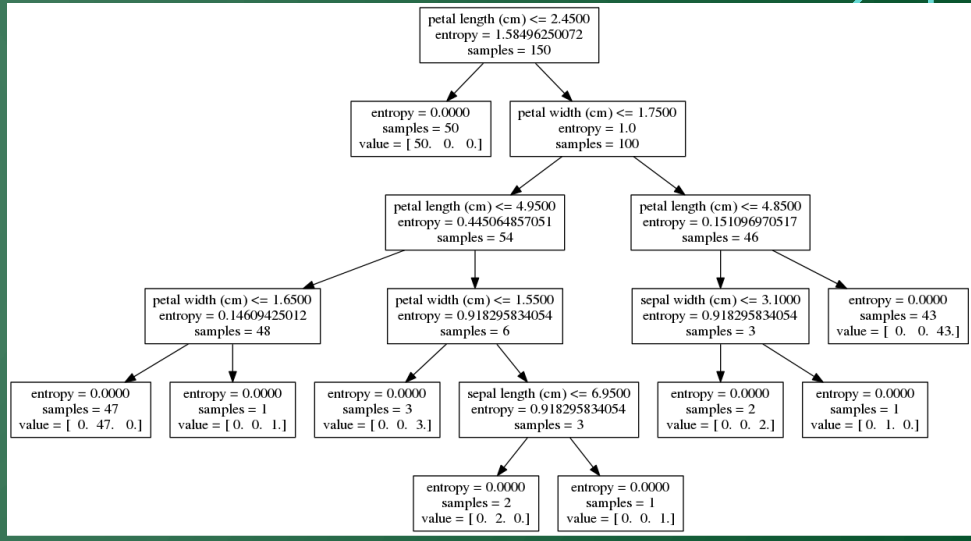
- Uses regression with a Sigmoid Function to estimate Forecast Probabilities:

$$FP_{yx}(LogReg) = \frac{e^{\boldsymbol{\beta}_{(yx)}\boldsymbol{X}_{yx}}}{1 + e^{\boldsymbol{\beta}_{(yx)}\boldsymbol{X}_{yx}}}$$

Algorithm	Linear Regression	Logistic Regression
Distribution Name	Gaussian Distribution	Bernoulli Distribution
Distribution Function	$f(x\|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$f(x\|p) = \begin{cases} p & x = 1 \\ 1-p & x = 0 \\ 0 & else \end{cases}$ $f(x\|p) = p\big(\delta(x-1)\big) + (1-p)\big(\delta(x)\big)$
Linear Predictor	$\eta = X\beta$	$\eta = X\beta$
Link Name	Identity	Logit/Inverse Sigmoid
(Inverse) Link Function	$E[Y] = X\beta$	$E[Y] = \frac{e^{X\beta}}{1 + e^{X\beta}}$

Tree-based Approaches

Decision Trees



- Acts to make a categorical prediction by means of a conjunction of boolean variables derived from an example's feature vector
- Network of two types of nodes: *decision nodes* and *leaf nodes*.
- Decision nodes each have exactly two children, which may be either decision nodes or leaf nodes, with a binary split based on the numeric value of a single feature from the an input example's feature vector.
- A leaf node has no children and instead, makes a categorical prediction of the verifying class of the input example based on the leaf's relationship to its ancestor nodes.
- For a given training example, begin at a decision tree's *root*
- At each decision node, compare the value of its feature to the critical threshold of the corresponding feature prescribed for that decision node.
- If the example's feature exceeds the node's critical threshold, proceed to right child; otherwise, traverse to left child.
- Repeat until arrive at leaf node; predict verifying category associated with leaf at this point, the value corresponding to the leaf becomes the predicted verifying category for the input example.
- Tree Construction:
  - Suppose a decision tree is trained on *n* training examples, each with a feature vector F of length *m*. At a given node *k*, the candidate splits S consist of a feature f and threshold θ, $S = (f, \theta)$. The set of training examples that traverse the developing tree to reach k is denoted Q. S partitions Q into Q_{left} and Q_{right} by: $Q_{left} = \forall \langle y, F \rangle (F[f] < \theta)$; $Q_{right} = \forall \langle y, F \rangle (F[f] \geq \theta)$. There is said to be *impurity* I at k based on S; that is given by $I(Q,S) = \frac{len(Q_{left})}{len(Q)} H\left(Q_{left}(S)\right) + \frac{len(Q_{right})}{len(Q)} H\left(Q_{right}(S)\right)$, where H is the *impurity function*. Among the candidate splits S at k, the chosen split S* is the split satisfying: $S^* = argmin_S\big(I(Q,S)\big)$. This process of greedy split selection is continued recursively until the *termination criterion* is satisfied.
  - Traditionally, the *termination criterion* is simply that a node k is a *decision node* unless $len(Q) = 1$, in which case a leaf node with prediction y_0 ($Q = \langle y_o, F_o \rangle$). However, recursing this deep is very susceptible to fitting the noise of the training data, thereby *overfitting* the predictive model and degrading its generalized skill. To alleviate this concern, often more liberal termination criterion are applied, such as creating a leaf node whenever $len(Q) \leq len_{min}$; $len_{min} > 1$, or by imposing a maximum allowable depth D of the tree $depth(k) \leq D$.

Example Decision Tree

Synthetic Example
Predictors: PWAT, U10, and CAPE

Day	ARI exceeded?	PW (in.)	CAPE (J/kg)	U10 (m/s)
D1	Yes	1.7	2500	5.5
D2	Yes	0.8	2200	12.5
D3	No	0.4	0	18
D4	No	1.8	800	5
D5	No	1.3	1200	7
D6	No	1.8	200	9
D7	Yes	1.4	1600	13.5
D8	Yes	2.2	1800	8
D9	No	1.9	1000	14
D10	Yes	1.6	3600	5
D11	No	1.2	100	10
D12	Yes	1.7	2300	8
D13	Yes	1.6	1600	4
D14	No	1.0	3500	2

PW

<1.5”

Day	ARI exceeded?	PW	CAPE	U10
D2	Yes	<1.5	>1500	>7.5
D3	No	<1.5	<1500	>7.5
D5	No	<1.5	<1500	<7.5
D7	Yes	<1.5	>1500	>7.5
D11	No	<1.5	<1500	>7.5
D14	No	<1.5	>1500	<7.5

2 Yes, 4 No
Impure Node -> Keep going

≥1.5”

Day	ARI exceeded?	PW	CAPE	U10
D1	Yes	>1.5	>1500	<7.5
D4	No	>1.5	<1500	<7.5
D6	No	>1.5	<1500	>7.5
D8	Yes	>1.5	>1500	>7.5
D9	No	>1.5	<1500	>7.5
D10	Yes	>1.5	>1500	<7.5
D12	Yes	>1.5	>1500	>7.5
D13	Yes	>1.5	>1500	<7.5

5 Yes 3 No
Impure Node -> Keep going

CAPE

<1500

Day	ARI exc?	PW	CAPE	U10
D3	No	<1.5	<1500	>7.5
D5	No	<1.5	<1500	<7.5
D11	No	<1.5	<1500	>7.5

0 Yes, 3 No
Pure Node -> Stop
PREDICT NO

≥1500

Day	ARI exc?	PW	CAPE	U10
D2	Yes	<1.5	>1500	>7.5
D7	Yes	<1.5	>1500	>7.5
D14	No	<1.5	>1500	<7.5

2 Yes, 1 No
Impure Node -> Keep going

CAPE

<1500

Day	ARI exc?	PW	CAPE	U10
D4	No	>1.5	<1500	<7.5
D6	No	>1.5	<1500	>7.5
D9	No	>1.5	<1500	>7.5

0 Yes, 3 No
Pure Node -> Stop
PREDICT NO

≥1500

Day	ARI exc?	PW	CAPE	U10
D1	Yes	>1.5	>1500	<7.5
D8	Yes	>1.5	>1500	>7.5
D10	Yes	>1.5	>1500	<7.5
D12	Yes	>1.5	>1500	>7.5
D13	Yes	>1.5	>1500	<7.5

5 Yes, 0 No
Pure Node -> Stop
PREDICT YES

U10

<7.5

Day	ARI exc?	PW	CAPE	U10
D14	No	<1.5	>1500	<7.5

0 Yes, 1 No
Pure Node -> Stop
PREDICT NO

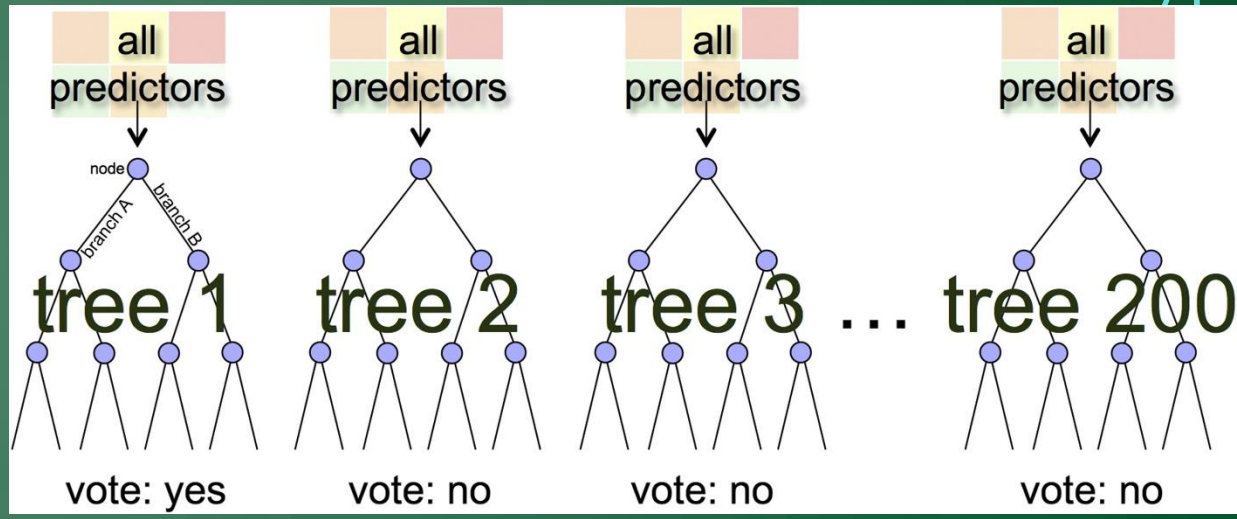
≥7.5

Day	ARI exc?	PW	CAPE	U10
D2	Yes	<1.5	>1500	>7.5
D7	Yes	<1.5	>1500	>7.5

2 Yes, 0 No
Pure Node -> Stop
PREDICT YES

Credit: Diana Stovern, WPC

Random Forests



Ahijevych et al. (2016)

- Decision trees: low *bias*, high *variance* solutions
  - Because few assumptions made, minimal error introduced by erroneous or oversimplistic assumptions in the model formulation
  - Very prone to *overfitting* the training data
  - Decision tree framework does not robustly extend to a probabilistic framework
    - Leaf nodes make deterministic predictions based on the mode verifying category of corresponding training data subset
- Random Forests Idea: use many different decision trees in tandem to form predictive model
  - Can significantly decrease the model *variance* with only a slight increase to the model *bias*, provided the trees are sufficiently uncorrelated.
- Challenge: how to generate a large set (forest) of reasonably skillful decision trees that are not strongly correlated?
  - DT forming algorithm is *deterministic*- it'll always form the same tree given the same training data
  - Two procedures for tree diversity: *tree bagging* and *feature bagging*.
    - Tree bagging: Simple resampling procedure. Form forest of size B from the *n* training examples by sampling, with replacement, *n* training examples from the original set B times; apply DT alg on samples.
    - Feature bagging: only a random subset of the *m* original input features are considered at each decision node

Tree/Forest Extensions

- A package called skgarden (https://scikit-garden.github.io/) offers a number of valuable extensions to decision trees and random forests
- Designed to be compatible with sklearn, skgarden classes have all the same functions, etc. as in sklearn
- Two general categories of extensions:

1. Different approaches to node splitting
   - Increase bias, decrease variance
   1. ExtraTrees (sklearn.tree.ExtraTree_; sklearn.ensemble.ExtraTrees_)
      - Like RF, but instead of picking the most discriminative threshold for a node split, randomly generate candidate split thresholds and choose among those
   2. Mondrian (skgarden.mondrian)
      - Online algorithm – don't need to process entire dataset in one batch
      - Feature, threshold for node split not based on target values at all. Splitting feature chosen randomly, weighted based on range of feature values (larger range -> more discriminative) and split threshold chosen randomly within the range of remaining values for the selected feature
2. Uncertainty quantification/accounting
   1. Quantile Regression Forests (skgarden.quantile)
      - Store information of all outcomes in leaf; thus have distribution of possible outcomes given the feature set
      - Can use to get prediction of sample mean from leaf of each tree activated by a particular forecast, but also quantiles (e.g. 90^{th} percentile) for a forecast
   2. Random Forests (skgarden.forest)
      - Instead of giving quantiles, give standard deviation of leaf values (which can be used to calculate quantiles by assuming normal distribution
      - Can be better than QRFs for assessing risk of very rare events

Some Other Algorithms

- Neural Networks
  - Will discuss these more after lunch
- K-Nearest Neighbors
- Boosting
- Support Vector Machines

Other Algorithms: K-Nearest Neighbors

- In sklearn.neighbors
- K-Nearest Neighbors
  - Pros:
    - No training phase
    - Very straightforward to understand and interpret
    - Easily extends to both Regression and Classification tasks
  - Cons:
    - Prediction can be (very) slow if the number of candidate neighbors is large (i.e. for large datasets)
    - Generally high variance model (i.e. prone to overfitting)
    - Typically generalization error will be higher than other models, though there are exceptions

Other Algorithms: Boosted Trees

- In sklearn.ensemble
- Boosting
  - Pros:
    - Generally lower variance than RFs and other algorithms
    - Tends not to have “failure modes” to the extent that some other methods have, since mispredicted examples increasingly weighted
  - Cons:
    - Cannot parallelize effectively
    - More parameter sensitivity/more difficult to tune than most algorithms
    - Can’t extrapolate beyond training example bounds

Other Algorithms: Support Vector Machines

- In sklearn.svm
- Support Vector Machines
  - Pros:
    - Generally has among the lowest generalization error among machine learning models
  - Cons:
    - Computationally expensive
    - Extensive parameter tuning required: several parameters, performance highly sensitive to each setting
    - Non-intuitive, more difficult to interpret

Model Summary

- Linear/Logistic Regression
  - Pros:
    - Efficient to train
    - Very interpretable*
  - Cons:
    - Imposed assumptions are often invalid
    - Can't effectively learn many relationships that are completely unknown to the developer prior to training
- Decision Trees
  - Non-parametric learning model;
  - Very useful construct, but use of single decision tree in isolation as predictive model is rarely if ever optimal approach
- Random Forests
  - Pros:
    - Very scalable
      - Highly parallelizable
      - Good computational complexity: Approximately linear in the number of training examples, square root in the number of predictors
    - Good for (multiclass) classification problems; easy extension to probabilistic prediction
    - Understandable and interpretable
    - Fewer tunable parameters and less sensitivity to parameter choices than other algorithms
  - Cons:
    - Can't extrapolate beyond training set
    - Still can't learn many sorts of predictor-predictand relationships; must be represented "step-wise)

Which algorithm should I use?

- It depends! No "one size fits all"
- Best to try out different approaches on your dataset(s) and compare
- Some important considerations:
  - Only consider algorithms adapted to the class of problem you are studying
    - Do you have observations for the phenomenon you're interested in studying/predicting? (i.e. is the problem supervised or unsupervised?)
    - Is the predictand continuous or discrete (e.g. regression or classification)?
  - Are the assumptions imposed by the algorithm valid?
    - Oftentimes you don't know
  - How much do you care about gleaning physical insight from the model vs. simply getting the best predictions?
  - Do you care about quantifying the uncertainty, or are you just interested in having the single best number?
  - How much training data do you have?