Machine Learning with scikits-learn

Make sure you have version 0.12 installed

```
easy_install -U scikit-learn
```

Joey Richards
jwrichar@stat.berkeley.edu

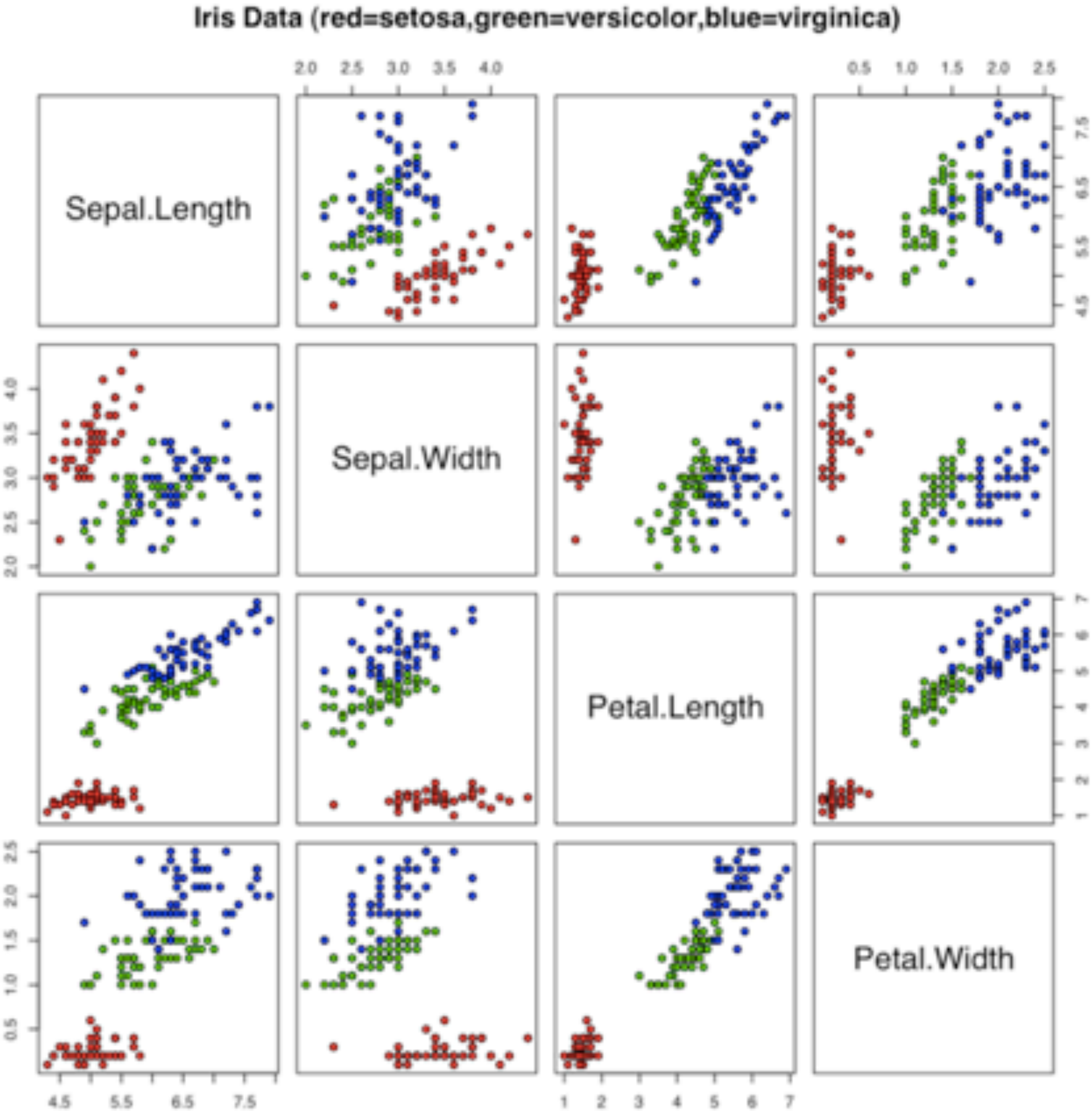
scikits-learn - Easy-to-use and general-purpose **machine learning** in Python

What is Machine Learning?

Short Answer: A cross between Statistics and Computer Science

Better Answer: A set of models which aim to learn something about a data set to apply that knowledge to new data

- Using labels from *training* data to classify new objects (e.g. images, digits, webpages)
- Learning the relationship between explanatory *features* and response variable to predict for new data (e.g. stock market)
- Discovering natural clustering structure in data
- Detecting low-dimensional structure in high-dimensional data
- Finding outliers in large data sets



setosa

veriscolor

virginica

Supervised Learning

Use *training* set of (***x***,y) pairs to learn to predict y for new ***x***

Regression - predicting continuous *outcome* (y) variable from a vector of input *features* (***x***)

- Linear Regression: `linear_model.LinearRegression`
- Lasso & Ridge Reg.: `linear_model.Lasso / linear_model.Ridge`
- Gaussian Process Regression: `gaussian_process.GaussianProcess`
- Nearest Neighbor Regression: `neighbors.KNeighborsRegressor`
- Support Vector Regression: `svm.SVR`
- Regression Trees: `tree.DecisionTreeRegressor`

... and more!

For more info on any of these methods, see:
http://scikit-learn.org/stable/user_guide.html

Supervised Learning, cont.

Classification - predicting the discrete class (y) of an object from a vector of input *features* (***x***)

- Logistic Regression: `linear_model.LogisticRegression`
- KNN Classification: `neighbors.KNeighborsClassifier`
- LDA / QDA: `lda.LDA / lda.QDA`
- Naive Bayes: `naive_bayes.GaussianNB`
- Support Vector Machines: `svm.SVC`
- Classification Trees: `tree.DecisionTreeClassifier`
- Random Forest: `ensemble.RandomForestClassifier`
- Multi-class & multi-label Classification is supported:

`multiclass.OneVsRest  multiclass.OneVsOne`

- **`feature_selection:`** recursive, L1, and tree-based

Model Fitting in scikits-learn

```
>>> from sklearn import datasets
>>> boston = datasets.load_boston() # Boston house-prices
>>> X = boston['data'] # 13 features (e.g. crime, # rooms, age, etc.)
>>> Y = boston['target'] # response (median house price)

# do linear regression
>>> from sklearn import linear_model
>>> clf = linear_model.LinearRegression()
# fit model on half of data
>>> half = floor(len(Y)/2)
>>> clf.fit(X[:half],Y[:half])
# predict for other half of data
>>> Y_lr_pred = clf.predict(X[half:])
>>> plot(Y[half:],Y_lr_pred - Y[half:],'o')

# KNN regression
>>> from sklearn import neighbors
>>> from sklearn import preprocessing
>>> X_scaled = preprocessing.scale(X) # many methods work better on scaled X
>>> clf1 = neighbors.KNeighborsRegressor(5)
>>> clf1.fit(X_scaled[:half],Y[:half])
>>> Y_knn_pred = clf1.predict(X_scaled[half:])
>>> plot(Y[half:], Y_knn_pred - Y[half:],'o')
```

Error Estimation & Model Selection

Q: How will our model perform on future data?

In the previous example, I split the data, using one set to ***train*** the model and the other to ***test*** its performance

scikits-learn can do cross-validation for us!

```
from sklearn import cross_validation
```

Better procedure:
Cross-Validation

K-fold CV - randomly split the training data into K folds. For each k=1,...,K, train model only on the data not in fold k & predict for data in fold k. Compute performance metric over CV predictions.

Leave-one-out (LOO) CV - K-fold CV with K = number of training points.

Error Estimation & Model Selection

Q: How do I choose which model and parameters to use?

KNN with what # of neighbors?
SVM which what kernel & bandwidth?
RF with how many trees and which mtry?
GP with what kernel & bandwidth?

Solution: use grid_search.GridSearchCV

`grid_search.`**`GridSearchCV`**(*estimator*, *param_grid*, *loss_func*, *n_jobs*, *cv=None*)

Computes *cv*-fold cross-validated *loss_func* (or *score_func*) of estimator over a *param_grid* on *n_jobs* cores, and returns the best model!

Error Estimation & Model Selection

Q: What evaluation metrics are available?

Loss Functions

metrics.zero_one(y_true, y_pred)
Zero-One classification loss
metrics.hinge_loss(y_true, pred_decision[, ...])
Cumulated hinge loss (non-regularized).
metrics.mean_square_error(y_true, y_pred)
Mean square error regression loss

Or write your own!!

Score Functions

metrics.zero_one_score(y_true, y_pred)
Zero-One classification score
metrics.auc(x, y)
Compute Area Under the Curve (AUC)
metrics.precision_score(y_true, y_pred[, ...])
Compute the precision
metrics.recall_score(y_true, y_pred[, pos_label])
Compute the recall
metrics.fbeta_score(y_true, y_pred, beta[, ...])
Compute fbeta score
metrics.f1_score(y_true, y_pred[, pos_label])
Compute f1 score

Evaluation Plots

metrics.confusion_matrix(y_true, y_pred[, ...]) Compute confusion matrix to evaluate the accuracy of a classification
metrics.roc_curve(y_true, y_score) Compute Receiver operating characteristic (ROC)
metrics.precision_recall_curve(y_true, ...) Compute precision-recall pairs for different probability thresholds

To the Notebook!

Unsupervised Learning

Clustering: K-means, Hierarchical Clustering, Mixture Models, Affinity Propogation, Spectral Clustering and lots of evaluation metrics!

Manifold Learning: Isomap, Local Linear Embedding, Hessian Eigenmap, Local Tangent Space Alignment

Matrix Factorization: PCA, Kernel PCA, Sparse PCA, ICA, NMF, Dictionary Learning

Outlier Detection: Elliptic envelope, One-class SVM

Covariance Estimation: Sparse, Shrunk, and Robust estimators

learn is missing a few things...

- Kernel smoothing / Loess
- Multivariate Adaptive Regression Splines (MARS)
- Boosting
- Multidimensional scaling (MDS)
- Neural Networks / Self-Organizing Maps
- Kalman Filtering / Particle Filtering
- Hidden Markov Models (not maintained)
- Missing data imputation / surrogate splitting
- Bayesian model fitting / non-parametrics

Breakout #2

Classify the famous **Iris** data, first used by R.A. Fisher.

To load in the data and split into training / testing sets:

```
iris = datasets.load_iris()
Xtr = iris.data[::2,:]
Xte = iris.data[1::2,:]
Ytr = iris.target[::2]
Yte = iris.target[1::2]
```

1. Choose your favorite classification model.
2. Find the parameters that maximize the 3-fold cross-validation 0-1 score function over the training set.
3. Apply this optimized model to predict the class of each object in the held-out set.

a) What is your best 3-fold CV 0-1 score?
b) What is your 0-1 score when applying it to testing set?