More Tools

Machine Learning

PHYS 453

Dr. Daugherity

More Tools

We need to discuss 3 more tools before we dig into how the individual classifiers work:

- **Pandas** – helps us deal with data
  - rapidly becoming obsolete but still in use everywhere so worth knowing
  - waiting for the world to agree on one standard replacement
- **Pipelines –** essential sklearn tool for building a consistent process of steps to manage data
- **Cross Validation** – vital tool for choosing and tuning a model

Big Picture

Typical steps in ML project:

- **Getting Data** – in the real world cleaning the data can be really difficult
- **Preprocessing** – getting data ready to use, only part we've discussed so far is test/train split
- **Training** – choosing models and parameters, "fitting" them to data
- **Evaluating** – reporting how well it works
- **Predicting** – now that you've got a working thing, go out and use it!

Step 1

GETTING DATA

Getting Data

- Can be anywhere from easy to agonizing
- Very problem specific
- Most generic tool is https://pandas.pydata.org/, community is working on faster replacements but no clear winner yet
- Learn how to clean and plot data!
- Seriously. Assume your data is messy until proven otherwise.
- Many techniques for dealing with missing/messy data are in sklearn, pandas, and elsewhere

Summarize Data

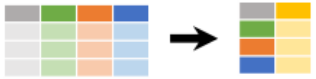
`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df.shape`
Tuple of # of rows, # of columns in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive and statistics for each column (or GroupBy).



pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

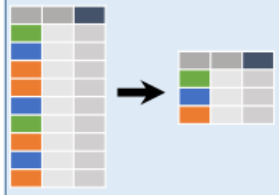
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

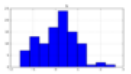
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

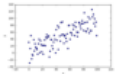
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Cheatsheet for pandas (http://pandas.pydata.org/) originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatshee

```
1 df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
1 df.dtypes  # check for non-numeric columns
```

```
sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
target                 int64
dtype: object
```

```
1 print('Number of NaN/null values:')
2 pd.isna(df).sum()
```

```
Number of NaN/null values:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

- check the stats, do you understand your data?
- obvious outliers?
- make sure dtypes are what you expect (columns which should be numeric will load as objects if one cell has text in it, and this will break lots of things)
- missing data?

Exploratory Data Analysis

In 1977, John Tukey, one of the great statisticians and mathematicians of all time, published a book entitled *Exploratory Data Analysis*. In it, he laid out general principles on how researchers should handle their first encounters with their data, before formal statistical inference. Most of us spend a lot of time doing exploratory data analysis, or EDA, without really knowing it. Mostly, EDA involves a graphical exploration of a data set.

We start off with a few wise words from John Tukey himself.

Useful EDA advice from John Tukey

- "Exploratory data analysis can never be the whole story, but nothing else can serve as a foundation stone—as the first step."
- "In exploratory data analysis there can be no substitute for flexibility; for adapting what is calculated—and what we hope plotted—both to the needs of the situation and the clues that the data have already provided."
- "There is no excuse for failing to plot and look."
- "There is often no substitute for the detective's microscope—or for the enlarging graphs."
- "Graphs force us to note the unexpected; nothing could be more important."
- "'Exploratory data analysis' is an attitude, a state of flexibility, a willingness to look for those things that we believe are not there, as well as those we believe to be there."

https://bebi103a.github.io/lessons/06/exploratory_data_analysis.html

Step 2

PREPROCESSING DATA

Preprocessing Data

Optimizing the data for use, sometimes called feature or data engineering. Examples:

- feature extraction: going from raw data to useful features
- encoding the data in a more efficient way
- combining features to make new ones
- eliminating unnecessary data

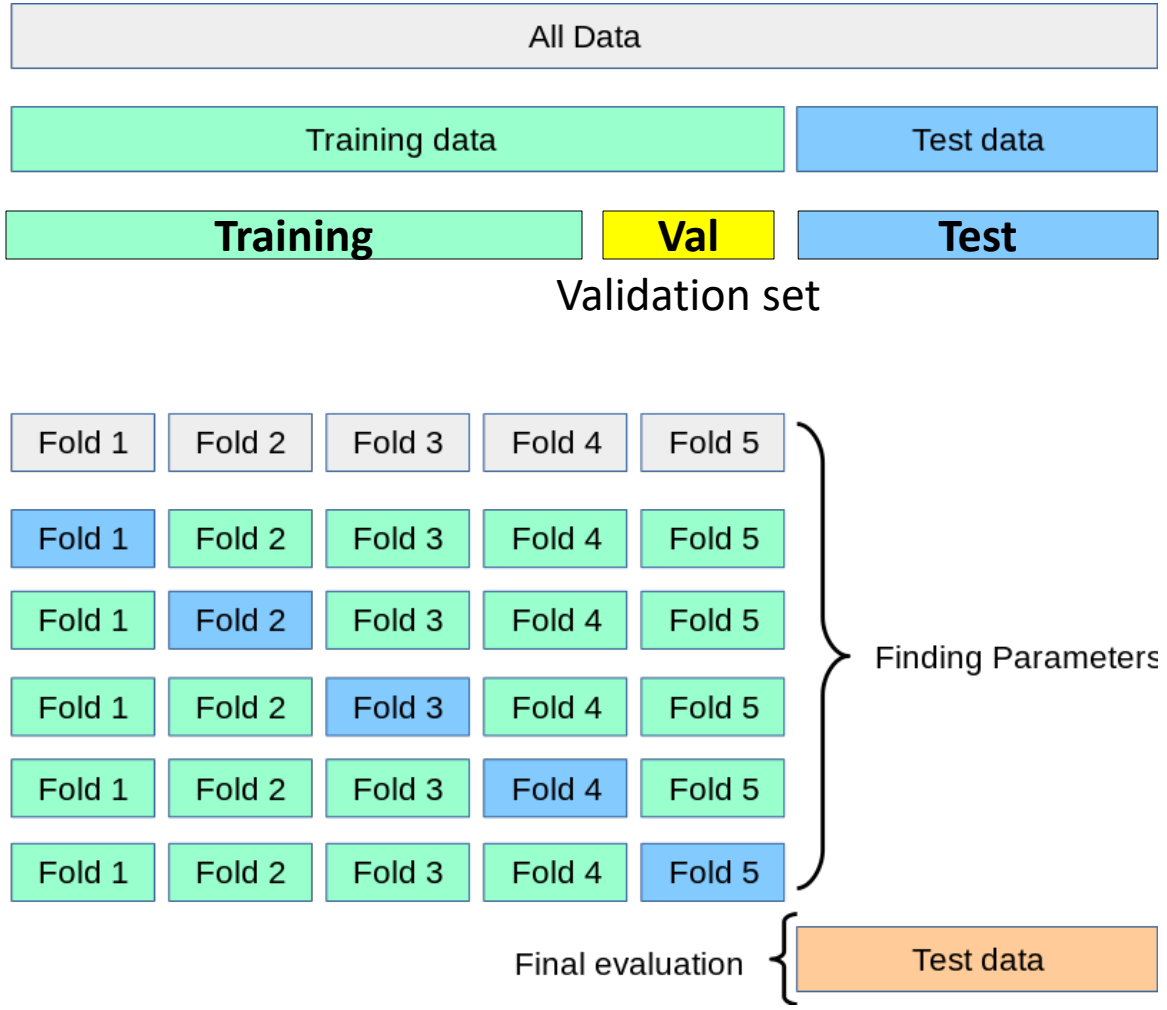
Can be done in sklearn, pandas, or something else

Test/Train Split

Crucial reminder: **you can NOT fairly evaluate with data you trained on!**

We will soon see how the training step requires trying lots of different options. We need to choose the best one, but we can't look at our test data yet.

We will set aside a "test set of the training set" called the **VALIDATION SET** to choose model and tune parameters.



https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

train_test_split

sklearn.model_selection.train_test_split(**arrays*, *test_size=None*, *train_size=None*, *random_state=None*, *shuffle=True*, *stratify=None*) [source]

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation, `next(ShuffleSplit().split(X, y))`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner.

Read more in the User Guide.

Parameters:

***arrays** : *sequence of indexables with same length / shape[0]*

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : *int, RandomState instance or None, default=None*

Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : *bool, default=True*

Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

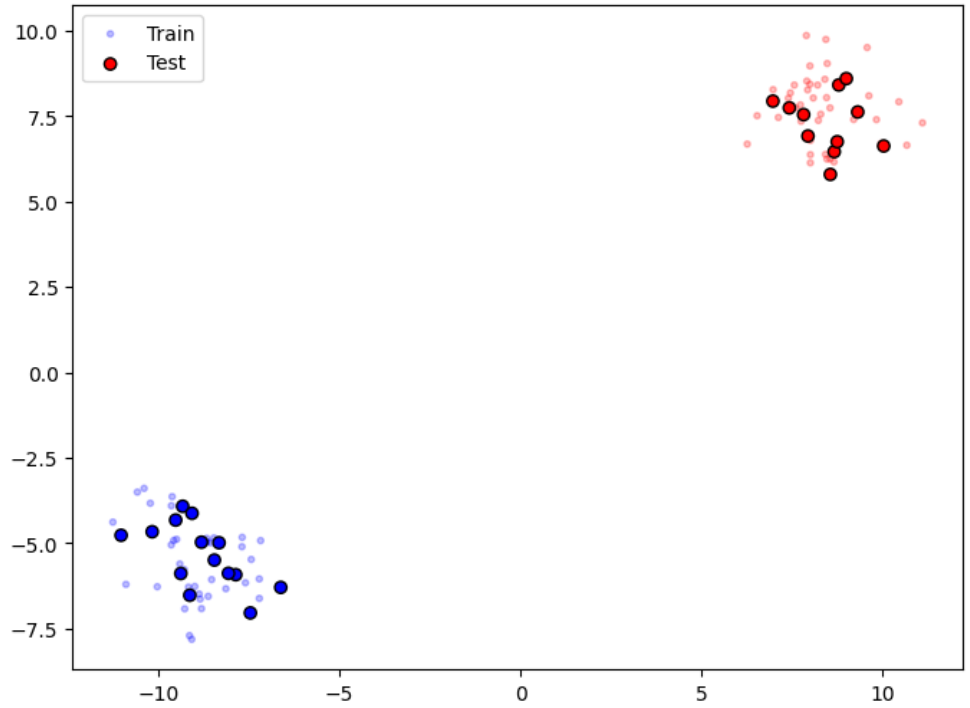
stratify : *array-like, default=None*

If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
print(y.shape, y_train.shape, y_test.shape)

plt.figure(figsize=(8,6))
plt.scatter(X_train[:,0], X_train[:,1],marker='.',c=y_train,cmap='bwr',label='Train', alpha=0.25)
plt.scatter(X_test[:,0], X_test[:,1],marker='o',edgecolor='k',c=y_test,cmap='bwr',label='Test')
plt.legend()

plt.show()
```



Feature Scaling

Consider a dataset that has features with very different scales.

- x1 from [0.01 to 0.04]
- x2 from [20,000 to 30,000]

- Some classifiers don’t care (decision trees)
- But some will behave oddly…
- Nearest Neighbors will effectively ignore x1
- Neural Networks will have a hard time converging

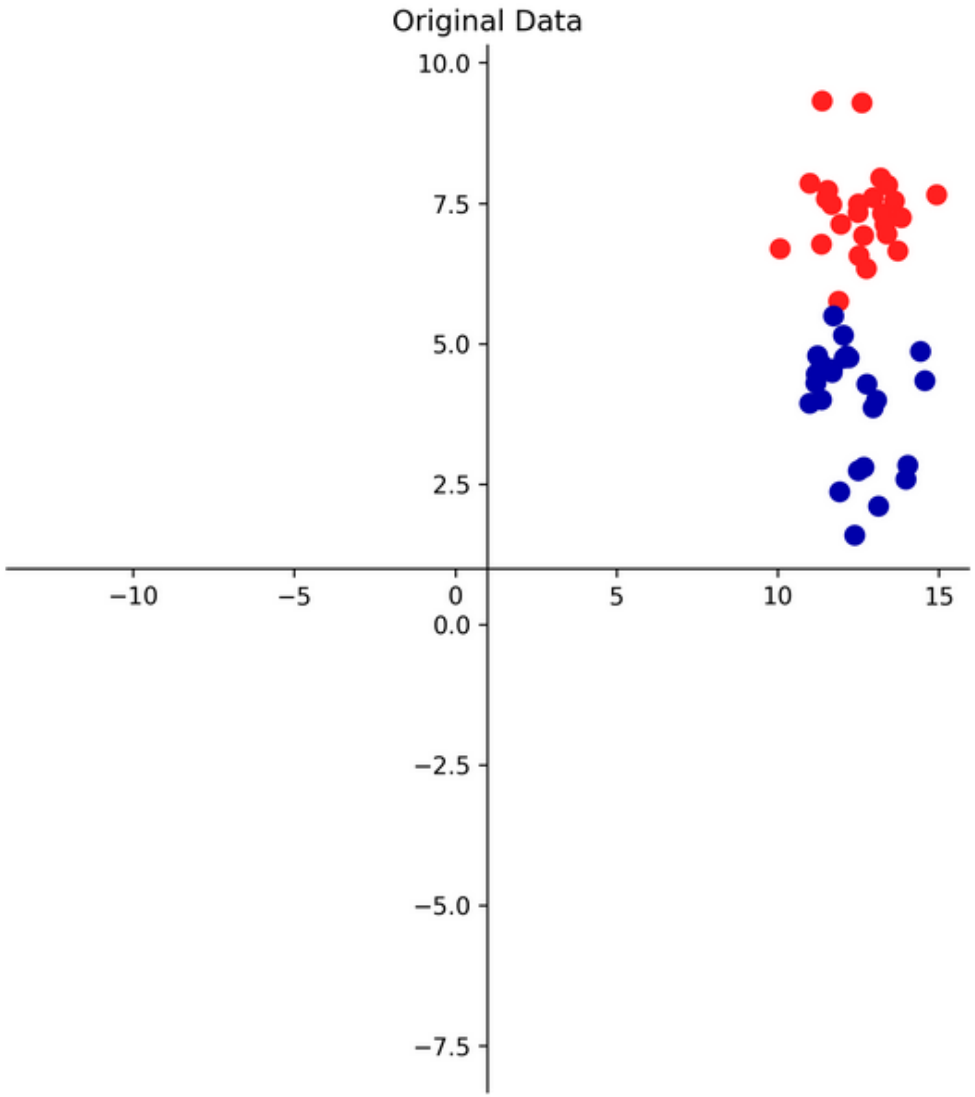
Feature Scaling

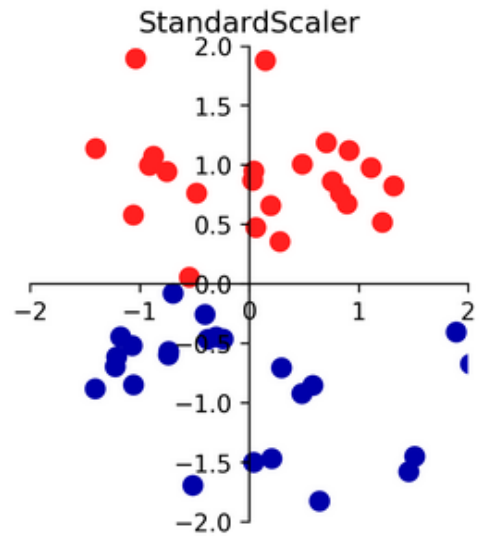
- It is always a good idea to scale your input data
- Warning: only fit the scalers to the training data, but make sure scaling gets applied to all data equally

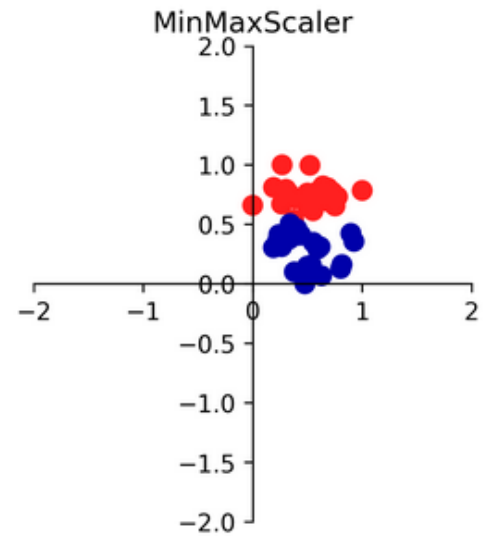
References

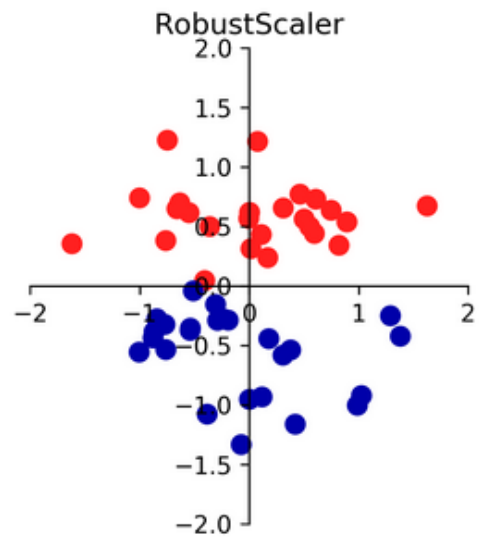
- User's Guide http://scikit-learn.org/stable/modules/preprocessing.html
- StandardScaler class: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- Intro to ML Chapter 3: https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb

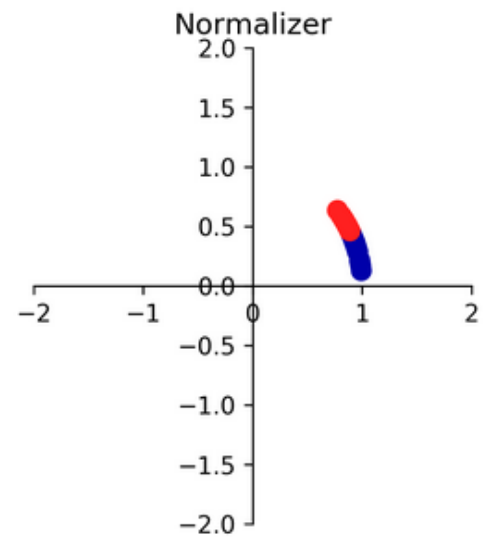
https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb



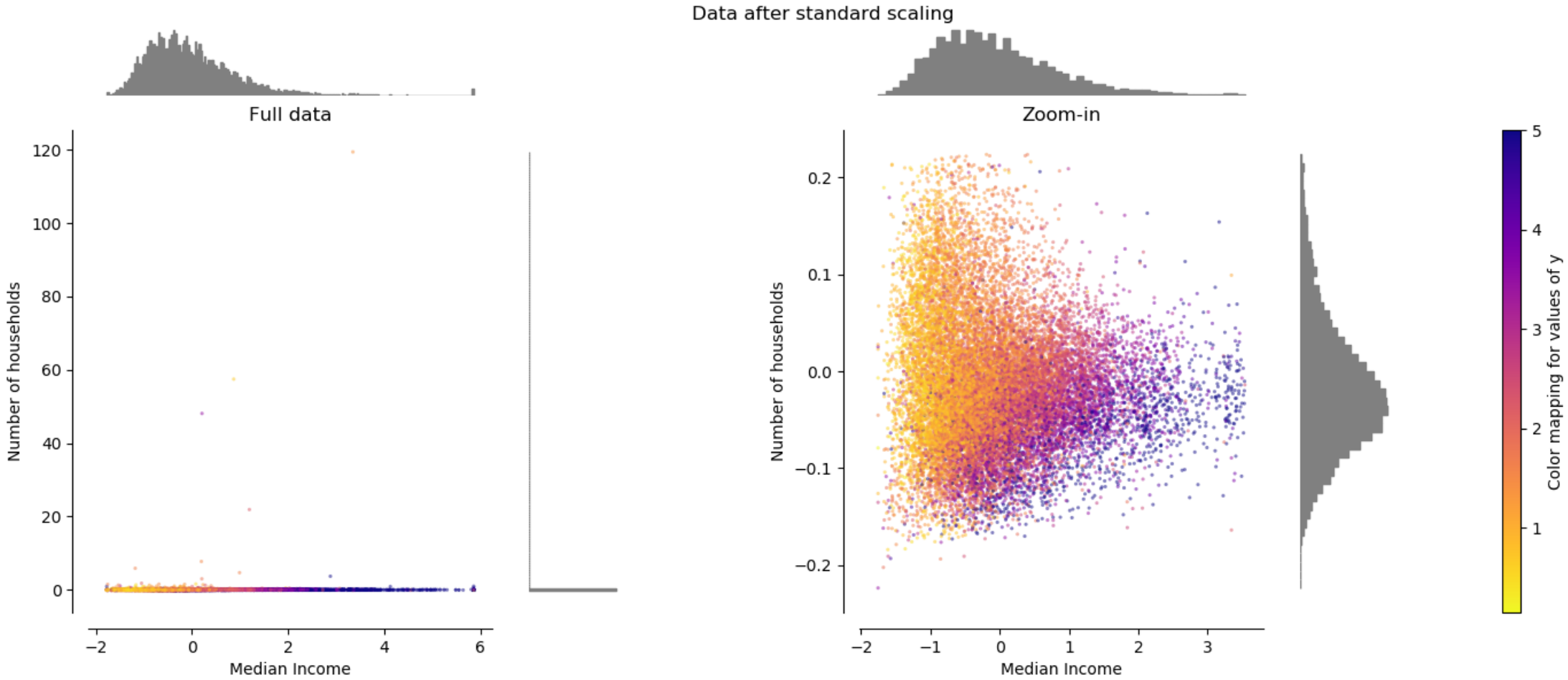








http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html



http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler

StandardScaler will set the mean to zero and variance to one

Methods

`fit` (X[, y])	Compute the mean and std to be used for later scaling.
`fit_transform` (X[, y])	Fit to data, then transform it.
`get_params` ([deep])	Get parameters for this estimator.
`inverse_transform` (X[, copy])	Scale back the data to the original representation
`partial_fit` (X[, y])	Online computation of mean and std on X for later scaling.
`set_params` (**params)	Set the parameters of this estimator.
`transform` (X[, y, copy])	Perform standardization by centering and scaling

BE CAREFUL to transform all data the same way!

- don’t re-fit your testing data
- don’t forget to transform testing data

Pipelines

Warning on a huge potential bug!

```
# BUGGY CODE
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # NEVER FIT TO TEST DATA!
clf.fit(X_train_scaled,y_train)
print('SCALED score:\t',clf.score(X_test,y_test)) # BUG: Forgot to scale test data!!!
```

We've now added a bunch of data processing steps, so we need a tool to ensure that all data gets treated consistently

Pipelines

We will build a **pipeline** to manage data:

- https://scikit-learn.org/stable/modules/compose.html#pipeline
- https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline

The last step in our pipeline can be a classifier!

The order matters:

1. test/train split first
2. build pipeline
3. fit pipeline to train
4. use pipeline to predict or score

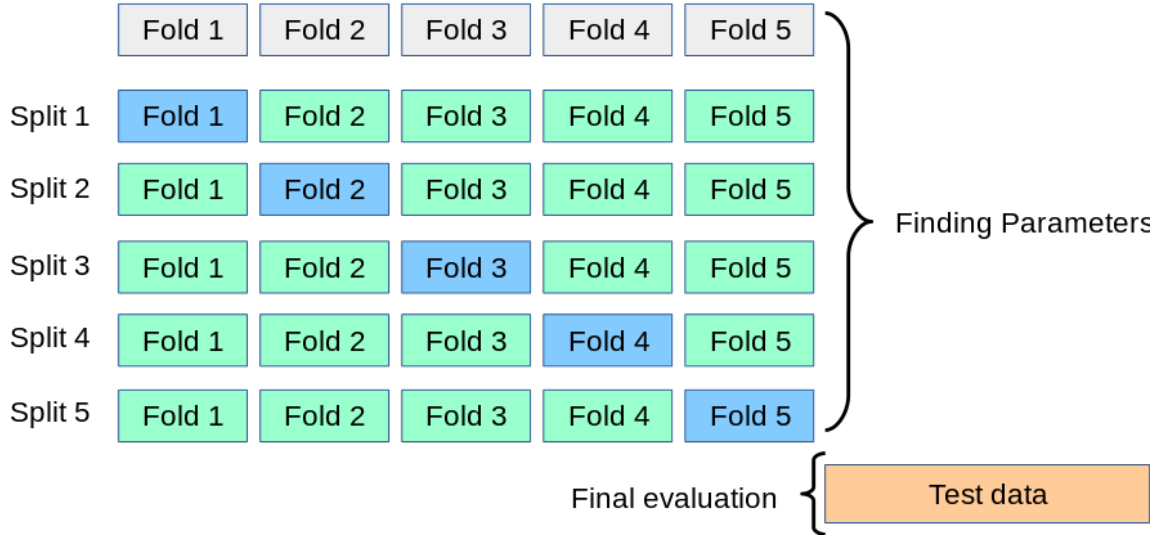
Step 3

TRAINING

Cross Validation

Problem: Need to choose which model to use and need to tune parameters

Solution: Cross Validation!



- helps us choose models and parameters first, then we only look at test set for final performance evaluation
- similar to test/train split: save a portion of training data called the validation set for scoring
- Folds: repeat process 5 times using different $1/5^{th}$ of data for validation, report final average
- Default Settings:
  - 25% of data is test
  - Split remaining 75% again: 80% of this is train, 20% is validation
  - Final totals: test=25%, train=.75*.8 = 60%, validation = 0.75*0.2 = 15%

Grid Search

- How do we tune parameters?
- Grid Search with Cross Validation!
- Brute force try every single combination and pick highest validation score
- Can get out of hand, but for big problems other methods exist that don’t try every possibility

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

GridSearchCV

class **sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, *, *scoring*=*None*, *n_jobs*=*None*, *refit*=*True*, *cv*=*None*, *verbose*=*0*, *pre_dispatch*='2*n_jobs', *error_score*=*nan*, *return_train_score*=*False*)** [source]

Exhaustive search over specified parameter values for an estimator.

Important members are fit, predict.

GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Read more in the User Guide.

Parameters:

estimator : ***estimator object***

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : ***dict or list of dictionaries***

Dictionary with parameters names (`str`) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

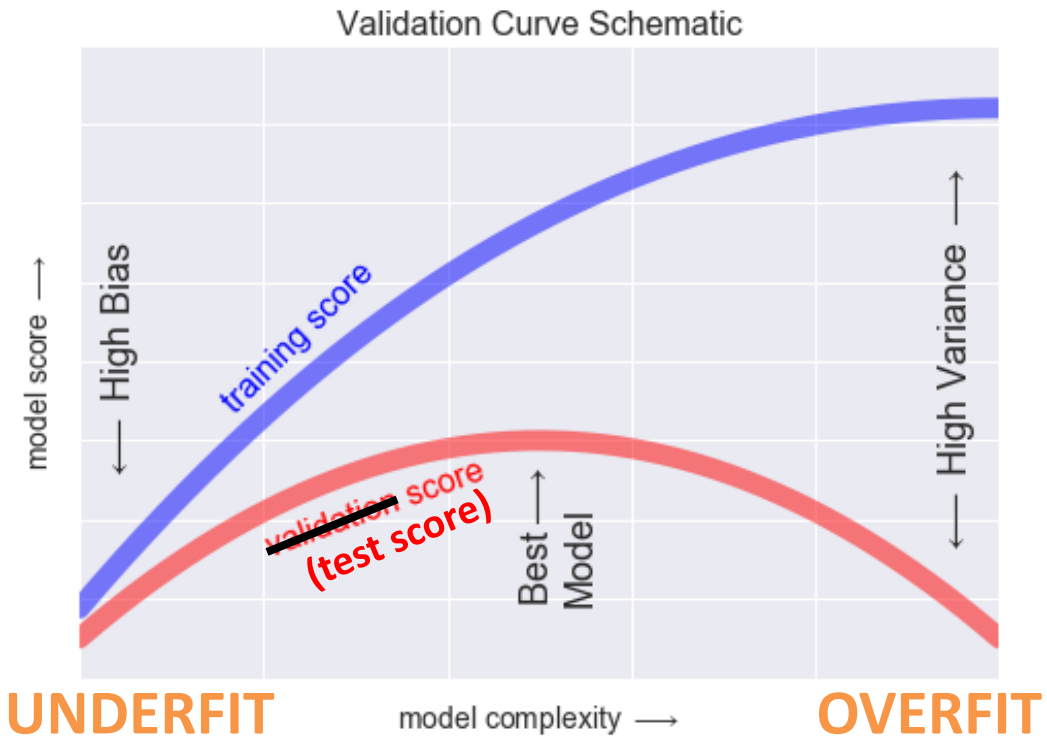
```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC()
>>> clf = GridSearchCV(svc, parameters)
>>> clf.fit(iris.data, iris.target)
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})
>>> sorted(clf.cv_results_.keys())
['mean_fit_time', 'mean_score_time', 'mean_test_score',...
 'param_C', 'param_kernel', 'params',...
 'rank_test_score', 'split0_test_score',...
 'split2_test_score', ...
 'std_fit_time', 'std_score_time', 'std_test_score']
```

Test/Train Split

Compare training to validation scores to tune parameters and choose models. Save test set for very end.

	Validation Bad ~~Testing~~ Score	Validation Good ~~Testing~~ Score
Bad Training Score	UNDERFITTING High Bias / Low Variance	Generally impossible
Good Training Score	OVERFITTING Low Bias / High Variance	It works!

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb



•The training score is everywhere higher than the validation score. This is generally the case: the model will be a better fit to data it has seen than to data it has not seen.

•Very **low** model complexity: the training data is **under-fit**, poor predictor both for the training data and for any previously unseen data.

•Very **high** model complexity: the training data is **over-fit**, model predicts the training data very well but fails for any previously unseen data.

•For some intermediate value, the validation curve has a maximum at best trade-off

Step 4

EVALUATING

Evaluating

- AFTER you have chosen your model and parameters, then you can score it on the test data ONE TIME to know how it does on new data
- Metrics: accuracy/error, precision/recall, etc
- Confusion Matrix
- Use same pipeline on test data

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

sklearn.metrics.confusion_matrix

`sklearn.metrics.confusion_matrix`(*y_true, y_pred, *, labels=None, sample_weight=None, normalize=None*) [source]

Compute confusion matrix to evaluate the accuracy of a classification.

By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j.

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

Read more in the User Guide.

Parameters:

y_true : *array-like of shape (n_samples,)*
Ground truth (correct) target values.

y_pred : *array-like of shape (n_samples,)*
Estimated targets as returned by a classifier.

labels : *array-like of shape (n_classes), default=None*
List of labels to index the matrix. This may be used to reorder or select a subset of labels. If `None` is given, those that appear at least once in `y_true` or `y_pred` are used in sorted order.

sample_weight : *array-like of shape (n_samples,), default=None*
Sample weights.

New in version 0.18.

normalize : *{'true', 'pred', 'all'}, default=None*
Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population. If None, confusion matrix will not be normalized.

```
>>> import matplotlib.pyplot as plt
>>> from sklearn.datasets import make_classification
>>> from sklearn.metrics import ConfusionMatrixDisplay
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.svm import SVC
>>> X, y = make_classification(random_state=0)
>>> X_train, X_test, y_train, y_test = train_test_split(
...         X, y, random_state=0)
>>> clf = SVC(random_state=0)
>>> clf.fit(X_train, y_train)
SVC(random_state=0)
>>> y_pred = clf.predict(X_test)
>>> ConfusionMatrixDisplay.from_predictions(
...    y_test, y_pred)
<...>
>>> plt.show()
```

```
>>> import matplotlib.pyplot as plt
>>> from sklearn.datasets import make_classification
>>> from sklearn.metrics import ConfusionMatrixDisplay
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.svm import SVC
>>> X, y = make_classification(random_state=0)
>>> X_train, X_test, y_train, y_test = train_test_split(
...         X, y, random_state=0)
>>> clf = SVC(random_state=0)
>>> clf.fit(X_train, y_train)
SVC(random_state=0)
>>> ConfusionMatrixDisplay.from_estimator(
...     clf, X_test, y_test)
<...>
>>> plt.show()
```

classification_report

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

```
sklearn.metrics.classification_report(y_true, y_pred, *, labels=None,
target_names=None, sample_weight=None, digits=2, output_dict=False,
zero_division='warn')
```

[source]

Build a text report showing the main classification metrics.

Read more in the User Guide.

Parameters:

y_true : *1d array-like, or label indicator array / sparse matrix*

Ground truth (correct) target values.

y_pred : *1d array-like, or label indicator array / sparse matrix*

Estimated targets as returned by a classifier.

labels : *array-like of shape (n_labels,), default=None*

Optional list of label indices to include in the report.

target_names : *array-like of shape (n_labels,), default=None*

Optional display names matching the labels (same order).

sample_weight : *array-like of shape (n_samples,), default=None*

Sample weights.

digits : *int, default=2*

Number of digits for formatting output floating point values. When `output_dict` is `True`, this will be ignored and the returned values will not be rounded.

output_dict : *bool, default=False*

If True, return output as dict.

Step 5

PREDICTING

Predicting

- Now that you've got a working model you can go out and use it. Predict new stuff!
- Use same pipeline on new data
- Specialists in deploying working models often called ML Engineers or ML Ops

Design Cycle

- Like all design work, this is an iterative process. Always may need to go back to previous steps