Analysis Steps

Machine Learning

PHYS 453 – Spring 2024

Dr. Daugherity

ABILENE CHRISTIAN UNIVERSITY

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!

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

```
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

Preprocessing Data

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

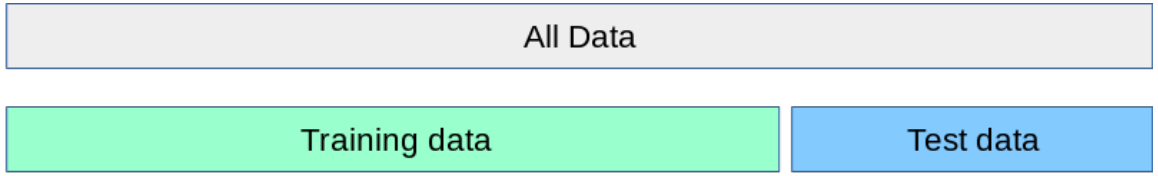
- feature extraction: going from raw data to useful features
- encoding the data in a more efficient way
- combining features to make new ones
- eliminating unnecessary data

Can be done in sklearn, pandas, or something else

Test/Train Split

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

Test/Train split: we save a subset of data where we know the right answer that we don't train on to evaluate performance

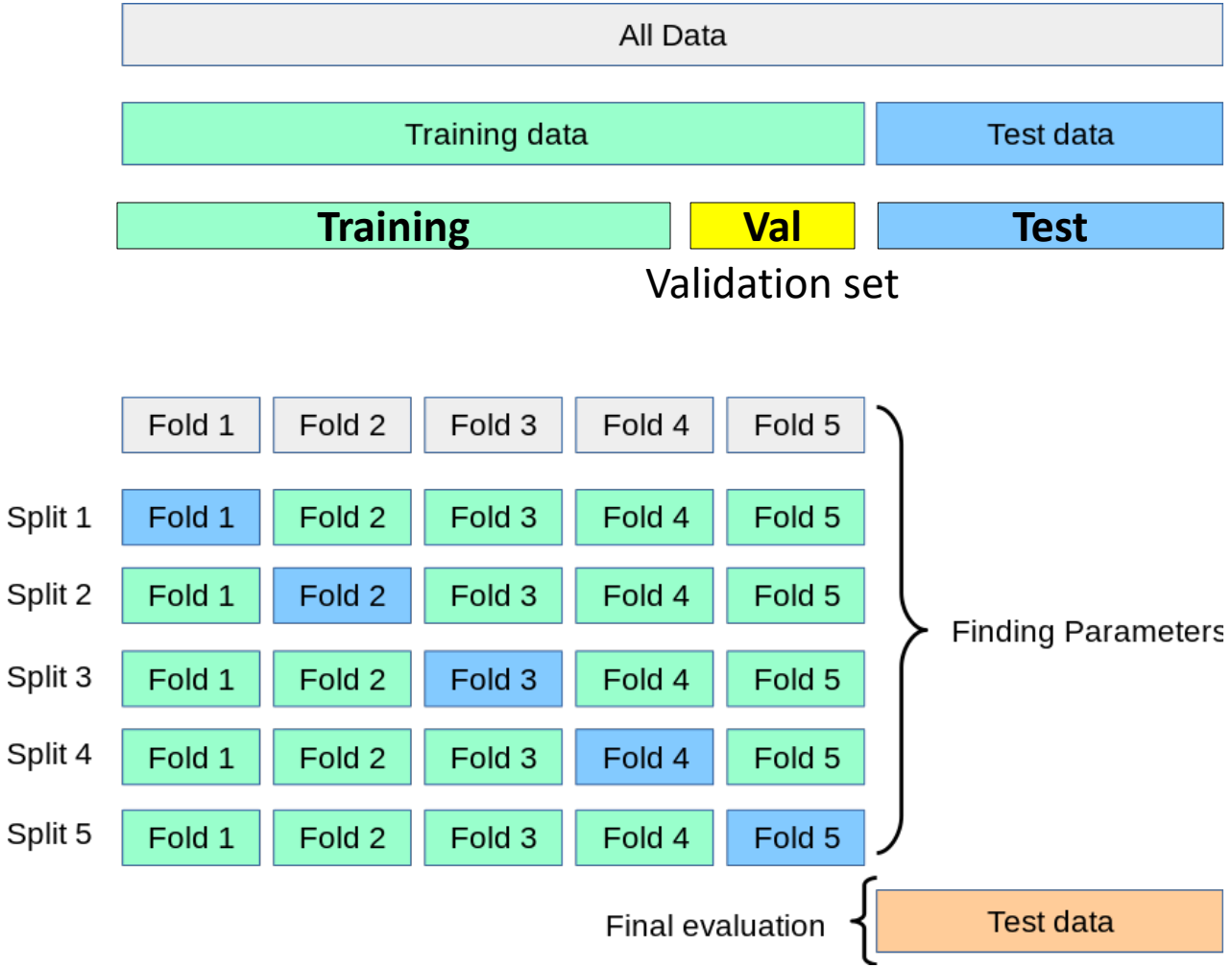


Test/Train Split

Vocab note: 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** which is different from the test set!

Later, we will use fancy cross-validation to make sure the validation is fair



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.

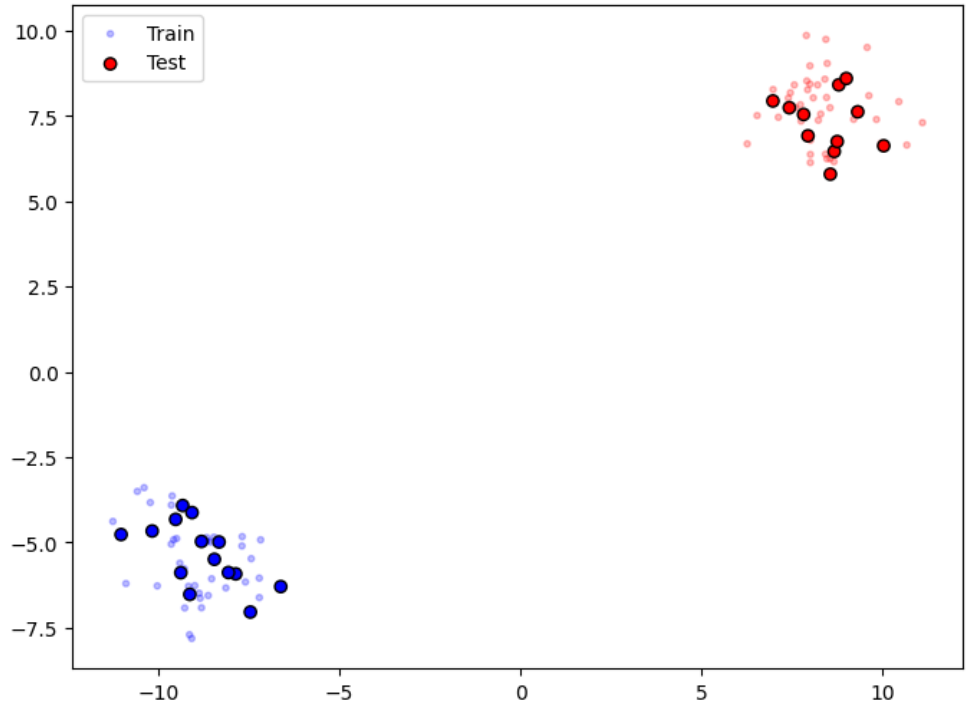
stratify : *array-like, default=None*

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()
```



Training

Easy interface in sklearn, for any classifier we call:

```
clf.fit(X_train, y_train)
```

- Here the classifier “learns” how to reproduce the output predictions from the input features.
- This can be as simple as finding a threshold as in HW1 or as complicated as finding billions of weights in a neural network.
- We can measure a training score

Never, ever, ever call: `clf.fit(X_test, y_test)`

No data leakage!

Training

- Model choices
- Every model has parameters that have to be set, how do we find the best ones?
- Grid Search with Cross Validation! Brute force try every single combination and pick highest validation score

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']
```

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

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.

Predicting

- Now that you've got a working model you can go out and use it. Predict new stuff!
- 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