Python and Machine Learning Day 4

By Craig Sakuma

Schedule - Day 4

Time	Topic
10:00 – 11:00	Review Data Cleaning and Matplotlib
11:00 – 12:00	Overview of Machine Learning
12:00 – 1:00	Lunch
1:00 – 3:30	K-Nearest Neighbors
3:30 – 3:45	Break
3:45 – 5:00	Random Forest

What is Machine Learning?

“A field of study that gives computers the ability to learn without being explicitly programmed.” (1959)

- Arthur Samuel, AI Pioneer

Netflix Prize

- Challenge to make 10% improvement in Netflix's recommendation system
- Grand prize was $1 million, with annual $50k progress prize to the leader at the end of the year
- 50k teams participated from over 180 countries
- Ratings matrix contained over 100 million numerical entries from 500k users across 17k movies
- Competition began in 2006 and the grand prize was awarded in 2009
- Winning entry was an ensemble of 100's of models

Supervised vs. Unsupervised

Supervised

- Requires truth set of data for training algorithms
- Examples:
  - Forecasting sales
  - Classifying spam

Unsupervised

- Autonomous algorithm that requires no training
- Examples:
  - Cluster analysis
  - Anomaly detection

Machine Learning Categories

	Continuous	Categorical
Supervised	Regression	Classification
Unsupervised	Dimension Reduction	Clustering

Supervised Training

- Training Set
  - Data used to create coefficients for model
  - For example, data set used to create a linear regression
- Test Set
  - Data used to measure performance of trained model

Training Set

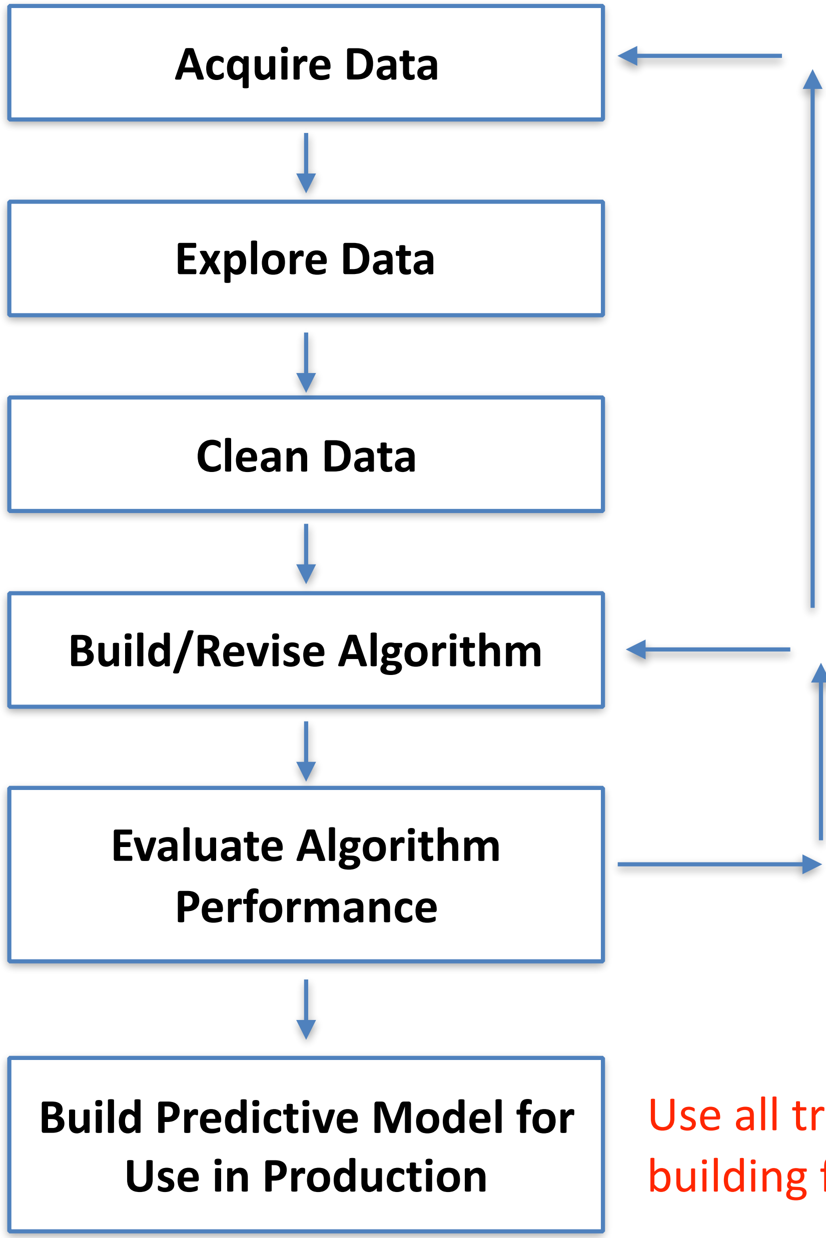
Age	*Sex*	*Pclass*	*Survived?*
25	*Male*	*3*	*FALSE*
17	*Female*	*1*	*TRUE*
40	*Male*	*2*	*FALSE*
9	*Female*	*2*	*TRUE*

Classes (target)

Independent Variables (features)

QuantSprout

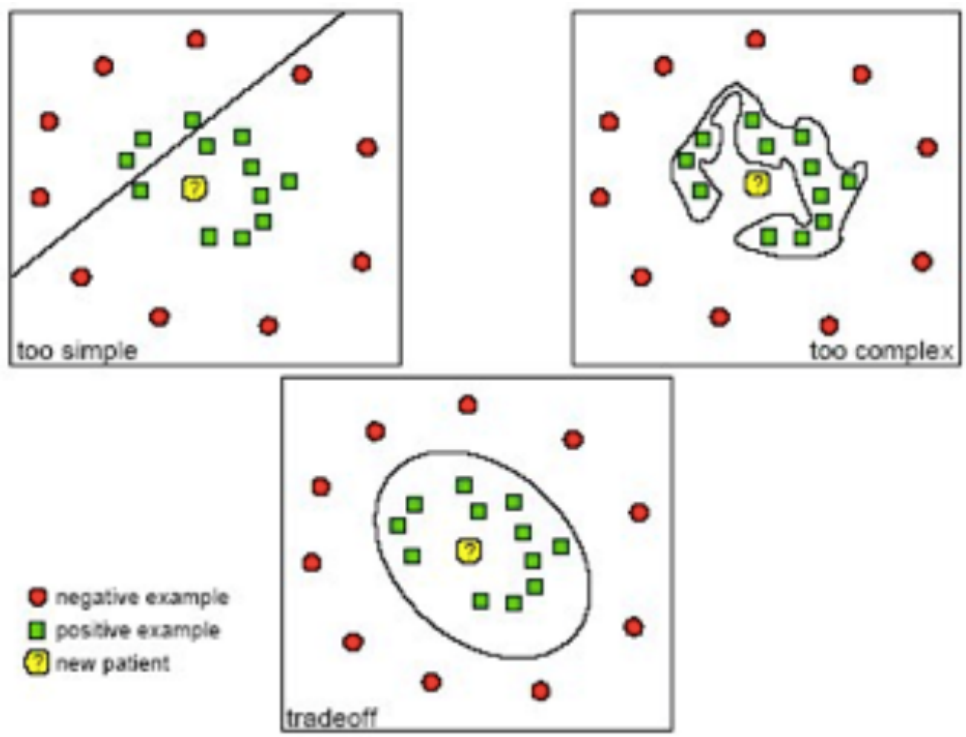
Data Science Process



Use all training data when building final model

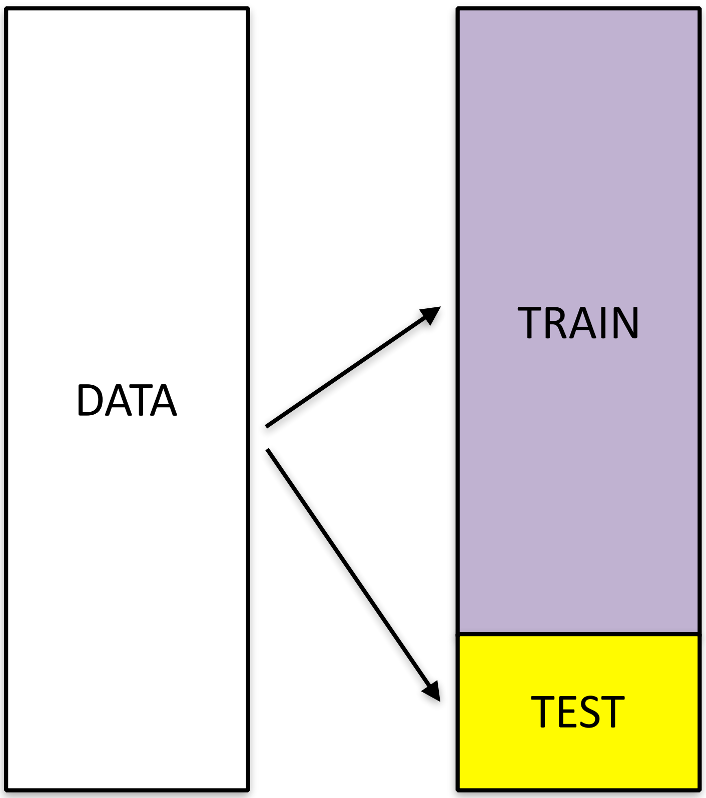
QuantSprout

Underfitting and Overfitting



Cross-validation

Split Data Set



- Separate the data into two groups
- Use part of data to train the model
- Use trained model to predict out come for test data
- Compare predictions with actual results

Measure model performance on accuracy of predictions

Iris Data Set

- Three species of iris
  - Setosa
  - Versicolour
  - Virginica
  - Species are encoded as 0,1 and 2 respectively
- Four measurements used to identify species
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width

Import the Iris Data Set

Import pandas and numpy packages

```
import numpy as np
import pandas as pd
```

Read the iris CSV file

```
iris = pd.read_csv('iris.csv')
iris.head()
```

Explore the data

```
print iris.info()
print iris.describe()
```

scikit learn

Home Installation Documentation Examples

Google Custom Search | Search ×

powered by Google

Previous sklearn.model... | Next sklearn.model... | Up API Reference

This documentation is for scikit-learn **version 0.18.1** — Other versions

If you use the software, please consider citing scikit-learn.

`sklearn.model_selection`.train_test_split

Examples using `sklearn.model_selection.train`

`sklearn.model_selection`.train_test_split

`sklearn.model_selection.` **`train_test_split`** (**arrays, **options*) ¶ [source]

Split arrays or matrices into random train and test subsets

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

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, int, or None (default is 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 automatically set to the complement of the train size. If train size is also None, test size is set to 0.25.

train_size : float, int, or None (default is 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 or RandomState

Pseudo-random number generator state used for random sampling.

stratify : array-like or None (default is None)

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

Returns:

splitting : list, length=2 * len(arrays)

Separate Features and Target

Create two variables for features and target.
Convert them into values.

```
iris_features = iris[['sepal_length', 'sepal_width',
            'petal_length', 'petal_width']].values
iris_target = iris.species.values
```

Scikit-learn has a function to split data into training and testing sets

```
from sklearn.model_selection import train_test_split
```

Train Test Split

Apply train_test_split to sample data

```
data = [['M',67],['Tu',74],['W',80],['Th',68],['F',78]]
sales = [240,360,700,320,1100]
```

```
f_train, f_test, t_train, t_test = \
            train_test_split(data,  sales,
            test_size=0.20,
            random_state=0)
```

QuantSprout

Train Test Split

View results of train_test_split

```
print 'Training Set'
print f_train
print t_train,'\n\n'
print 'Test Set'
print f_test
print t_test
```

Split Iris Data

Split the iris data into training set and test set

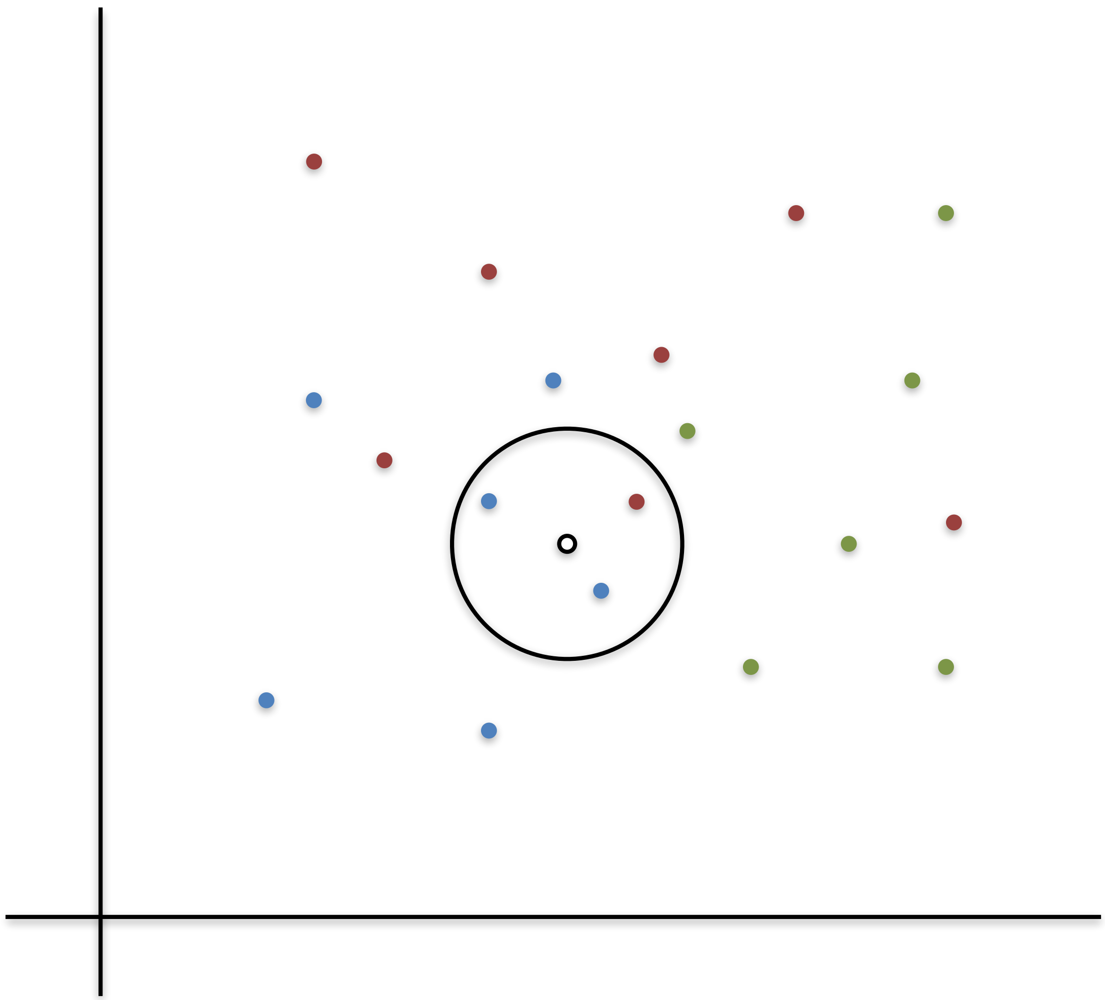
```
features_train, features_test, target_train, \
      target_test = train_test_split(
      iris_features, iris_target, test_size=0.20,
      random_state=0)
```

```
print features_train.shape
print features_test.shape
print target_train.shape
print target_test.shape
```

Classification Algorithms

- Logistic Regression
- Naive-Bayes
- Decision Trees
- Support Vector Machines
- Neural Networks
- K-Nearest Neighbors
- Random Forest

K Nearest Neighbors



Suppose you want to predict the white dot

1. Pick a value for k

2. Find colors of the k nearest neighbors

3. Assign the most common color

KNN Considerations

- Scaling of Data has large impact on algorithm (normalization is frequently used)
- Adjust the parameters of the algorithm
  - Number of neighbors
  - Treat data points uniformly or weighted by distance
- Can become computationally expensive at large scale

Home Installation Documentation Examples

Search

Previous
sklearn.neighbor...

Next
sklearn.neighbor...

Up
Reference

This documentation is for scikit-learn **version 0.16.1** — Other versions

If you use the software, please consider citing scikit-learn.

sklearn.neighbors.KNeighborsClassifier

Examples using `sklearn.neighbors.KNeighborsClassifier`

sklearn.neighbors.KNeighborsClassifier

class `sklearn.neighbors.`**`KNeighborsClassifier`**(*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, **kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Parameters:

n_neighbors : int, optional (default = 5)

Number of neighbors to use by default for `k_neighbors` queries.

weights : str or callable

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Uniform weights are used by default.

`fit(X, y)` [source]

Fit the model using X as training data and y as target values

Parameters:	
	X : {array-like, sparse matrix, BallTree, KDTree} Training data. If array or matrix, shape = [n_samples, n_features] **y** : {array-like, sparse matrix} Target values of shape = [n_samples] or [n_samples, n_outputs]

predict(*X*) [source]

Predict the class labels for the provided data

Parameters:	**X** : array of shape [n_samples, n_features] A 2-D array representing the test points.
Returns:	**y** : array of shape [n_samples] or [n_samples, n_outputs] Class labels for each data sample.

predict_proba(*X*) [source]

Return probability estimates for the test data X.

Parameters:	**X** : array, shape = (n_samples, n_features) A 2-D array representing the test points.
Returns:	**p** : array of shape = [n_samples, n_classes], or a list of n_outputs of such arrays if n_outputs > 1. The class probabilities of the input samples. Classes are ordered by lexicographic order.

score(*X, y, sample_weight=None*) [source]

Returns the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters:	**X** : array-like, shape = (n_samples, n_features) Test samples. **y** : array-like, shape = (n_samples) or (n_samples, n_outputs) True labels for X. **sample_weight** : array-like, shape = [n_samples], optional Sample weights.
Returns:	**score** : float Mean accuracy of self.predict(X) wrt. y.

K-Nearest Neighbors

Import K Nearest Neighbors from scikit-learn

```
from sklearn.neighbors import KNeighborsClassifier
```

Train the KNN classifier

```
model = KNeighborsClassifier(5).fit(features_train,
                                     target_train)
```

Compare predictions with actual results

```
print model.predict(features_test)
print target_test
```

K-Nearest Neighbors

Predict probabilities

```
model.predict_proba(features_test)
```

Score the model

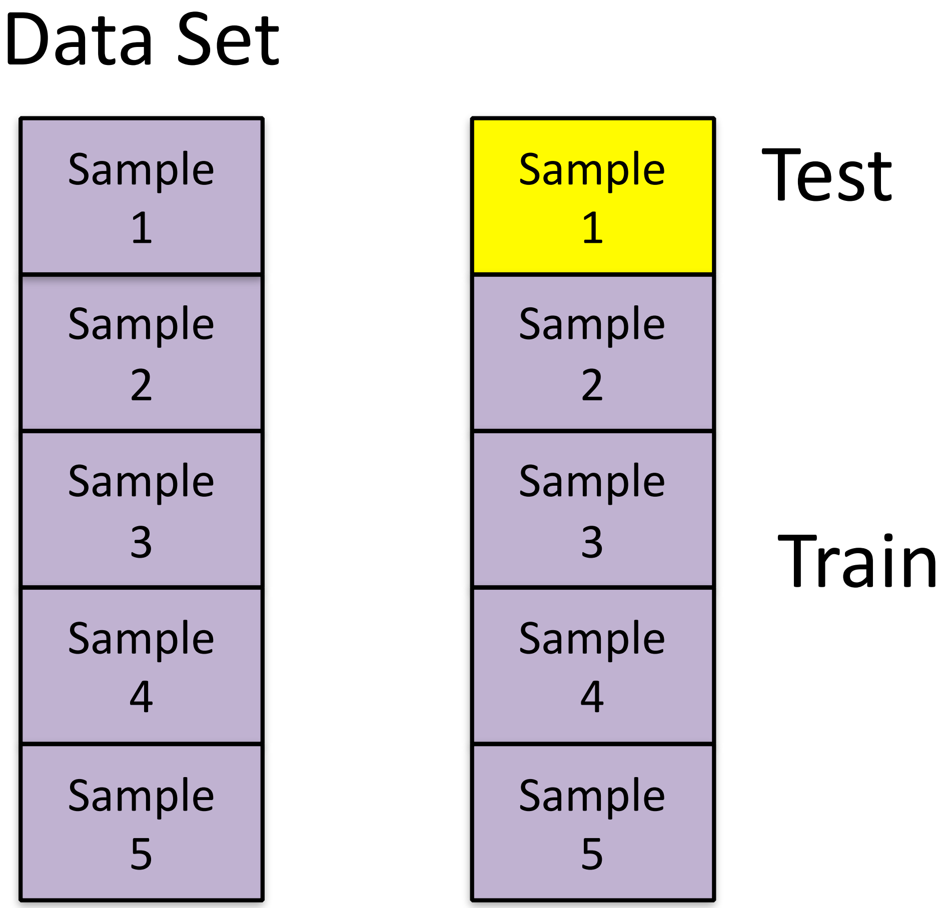
```
model.score(features_test, target_test)
```

Evaluating Algorithms

- Model performance will vary based on how you split the data into training and testing groups
- Taking an average of the model performance across different splits improves the measurement

KFold Cross-validation

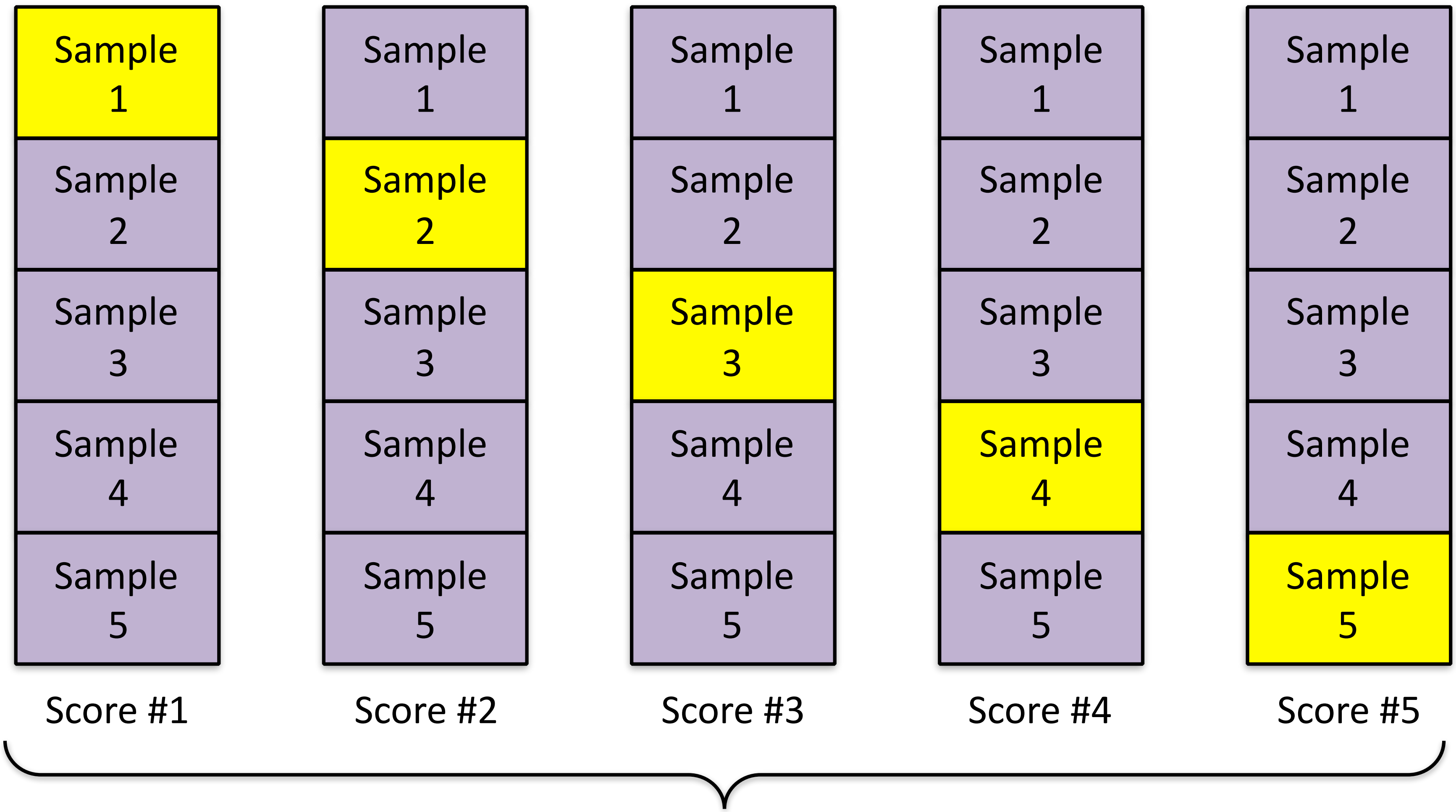
K-Folds Cross-validation



- Separate the data into K groups (e.g., 5)
- Each group maintains the same samples for all tests
- Train and test K times
- Sample group for test changes for each test

Every Data Point Is Tested Once

K-Folds Cross-validation



Average of Scores

scikit learn

Home Installation Documentation Examples

powered by Google

Google Custom Search Search ×

Previous sklearn.covar... Next sklearn.model... Up API Reference

This documentation is for scikit-learn **version 0.18.1** — Other versions

If you use the software, please consider citing scikit-learn.

`sklearn.model_selection`.KFold

Examples using `sklearn.model_selection.KFold`

`sklearn.model_selection`.KFold

class `sklearn.model_selection.` **`KFold`** (*n_splits=3*, *shuffle=False*, *random_state=None*) [source]

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

Parameters:	**n_splits** : int, default=3 Number of folds. Must be at least 2. **shuffle** : boolean, optional Whether to shuffle the data before splitting into batches. **random_state** : None, int or RandomState When shuffle=True, pseudo-random number generator state used for shuffling. If None, use default numpy RNG for shuffling.

`split` (*X, y=None, groups=None*) [source]

Generate indices to split data into training and test set.

Parameters:	**X** : array-like, shape (n_samples, n_features) Training data, where n_samples is the number of samples and n_features is the number of features. **y** : array-like, shape (n_samples,) The target variable for supervised learning problems. **groups** : array-like, with shape (n_samples,), optional Group labels for the samples used while splitting the dataset into train/test set.
Returns:	**train** : ndarray The training set indices for that split. **test** : ndarray The testing set indices for that split.

K-Folds Cross-Validation

Remember how to index numpy arrays with indices?

```
sample = np.array([0,10,20,30,40,50])
indices = [4,0,2]
print sample[indices]
```

K-Folds Cross-Validation

Import the KFold function from scikit-learn

```
from sklearn.model_selection import KFold
```

KFold function creates object for separating training and test sets. Lets create indices for a data set with 3 folds/splits.

```
k_fold_indices = KFold(3, shuffle=False)
print k_fold_indices
```

K-Folds Cross-Validation

Create some mock data

```
mock_features = np.array([[1,10],[2,20],[3,30]])
mock_target = np.array([100,200,300])
```

Let's see what is in the k_fold_indices

```
for train_indices,test_indices in \
k_fold_indices.split(mock_features,mock_target):
    print train_indices, test_indices
```

K-Folds Cross-Validation

Let's use indices to index values in an array.

```
for train_indices, test_indices in k_fold_indices.split(
        mock_features, mock_target):
    print 'Training Set'
    print mock_features[train_indices]
    print mock_target[train_indices]
    print 'Test Set'
    print mock_features[test_indices]
    print mock_target[test_indices]
    print
```

Cross-Validation Function

Inputs to function include data, classifier function, number of folds and random state

```
def cross_validate(features, target, classifier,
                   k_fold=10, r_state=None) :
```

Cross-Validation Function

Create sets of indices for separating data into training and test sets. There will be k sets of indices

```
k_fold_indices = KFold(k_fold,shuffle=True,
            random_state=r_state)
```

Cross-Validation Function

Iterate through the sets of indices to train and score each scenario

```
for train_indices, test_indices in k_fold_indices.split(
                    features, target):
    model = classifier.fit(features[train_indices],
                    target[train_indices])
    k_score = model.score(features[test_indices],
                    target[test_indices])
```

Use Cross-Validation Function

Test the function with different numbers of neighbors

```
print cross_validate(iris_features, iris_target,
                     KNeighborsClassifier(3), 10, 0)
print cross_validate(iris_features, iris_target,
                     KNeighborsClassifier(4), 10, 0)
print cross_validate(iris_features, iris_target,
                     KNeighborsClassifier(5), 10, 0)
```

Exercise

- Load the clean_data.csv file as DataFrame
- Separate the data into features (age, sex, pclass dummies) and target (survived) and convert to numpy array
- Evaluate KNN model using cross-validation function
- **Bonus:** Try different numbers of neighbors using a for loop

Data Normalization

- Technique for adjusting the scale of data
- Calculate mean and standard deviation of all samples for each variable
- Subtract the mean and divide by the standard deviation

Data Normalization

Calculate the mean and standard deviation

```
avg_age = data.Age_all.mean()
stdev_age = data.Age_all.std()
```

Subtract the mean and divide by the standard deviation

```
data['Age_norm'] = (
            data.Age_all - avg_age) / stdev_age
```

Data Normalization

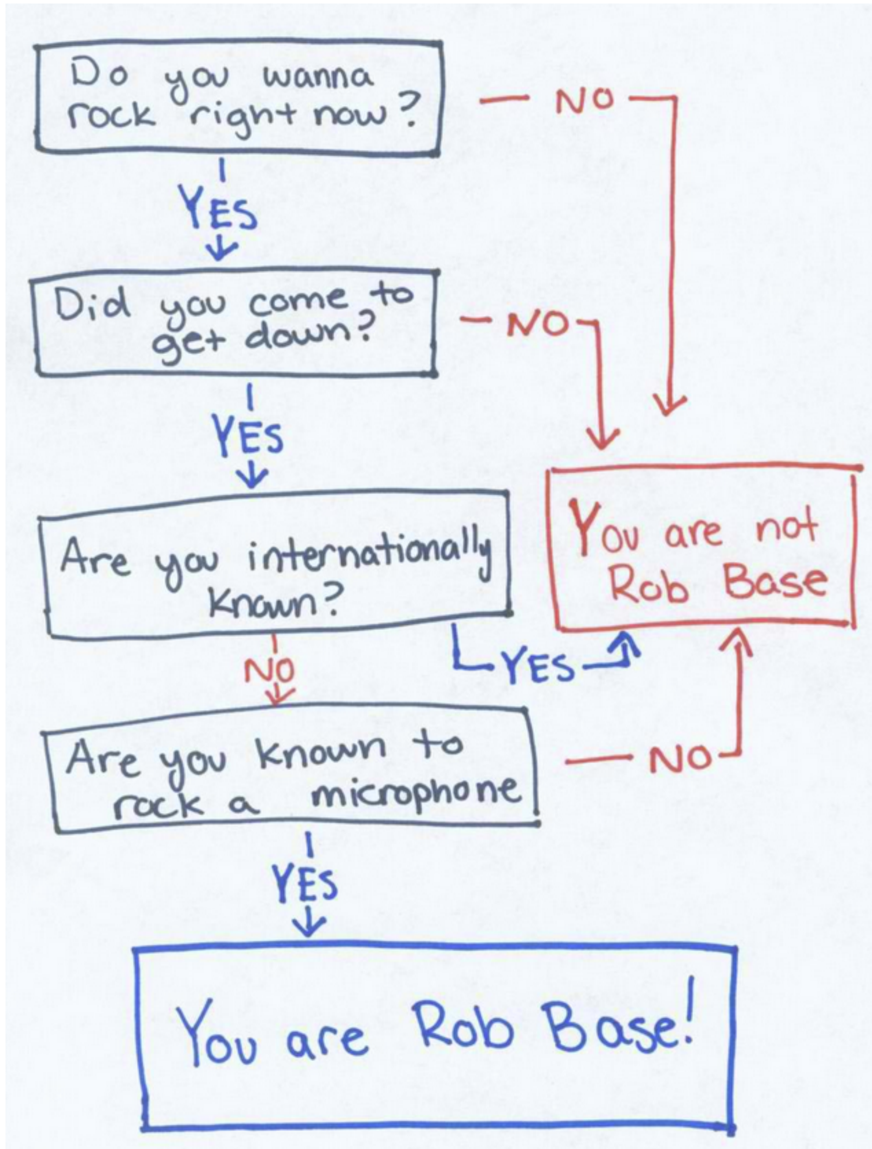
Create new features using normalized data

```
features_norm = data[['Age_norm', 'Sex', 'Pclass_1',
                                   'Pclass_2', 'Pclass_3']].values
```

Test KNN on normalized data

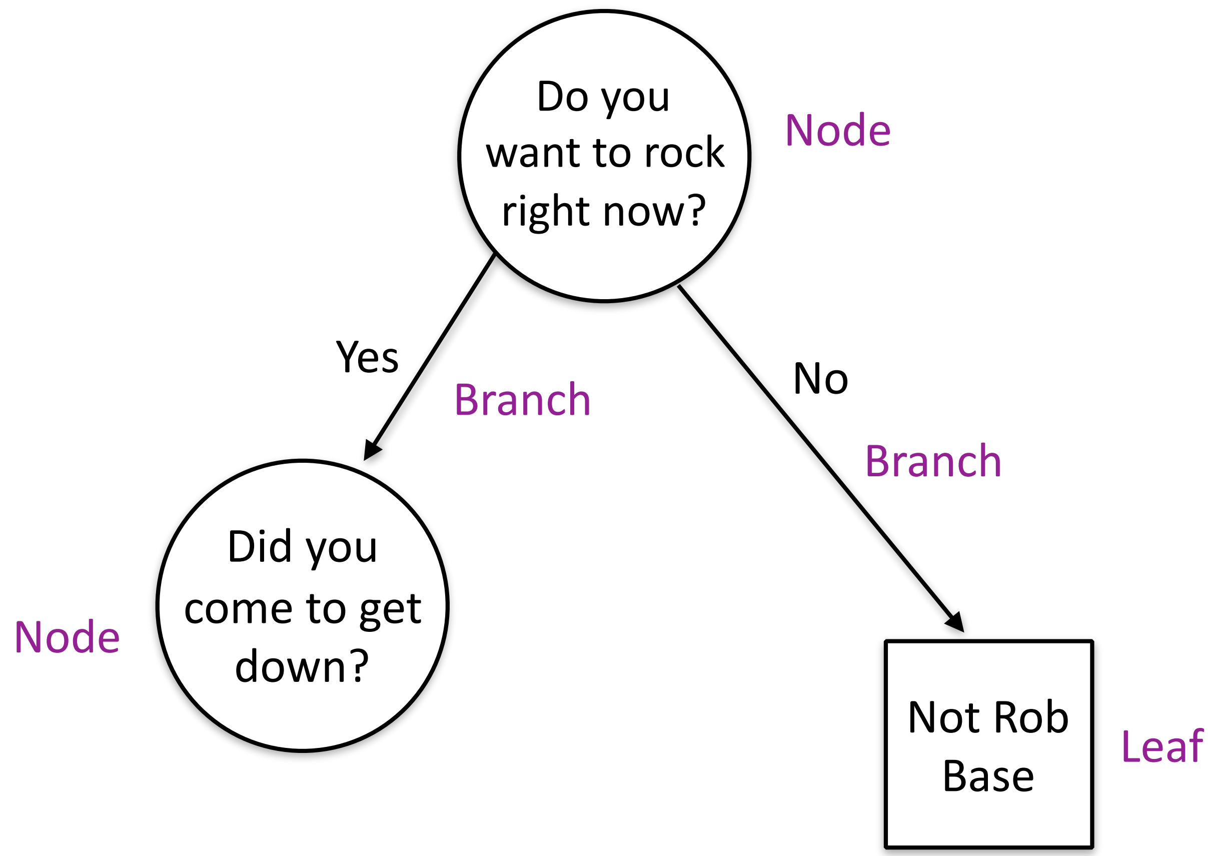
```
for k in range(1,10):
    model = KNeighborsClassifier(k)
    print cross_validate(features_norm, target,
                         model, 10, 0)
```

Is This a Decision Tree?



Decision Trees

- Trees consist of:
  - Nodes (questions)
  - Branches (answers)
  - Leaves (end points)
- Acyclic - flows in one direction
- No split is necessary when all records are from same class



Decision Trees

- Algorithm selects optimal node that creates the largest increase in purity
- Decision trees are susceptible to overfitting
- Techniques for preventing overfitting
    - Minimum number of records for leaf
    - Maximum depth for branches
- One technique is to purposely overfit, but manually prune branches

Random Forest

- Ensemble algorithm (mix of many models)
- Collection of decision trees
- Evaluates predictions from many models and selects the most common classification
- Features are randomly selected for each decision tree
- Bagging (Bootstrap Aggregating) is also applied to each tree
  - Sample of the data set is used for training
  - Samples are drawn with replacement

Why Random Forest Is Popular?

- Add all your data and algorithm will prioritize
- Not susceptible to overfitting
- Doesn’t require normalization
- Solid performance in wide range of applications

scikit learn

Home | Installation | Documentation | Examples

Google™ Custom Search | Search ×

Previous 3.2.4.3.6. sklea... | Next sklearn.ensemble... | Up Reference

This documentation is for scikit-learn **version 0.16.1** — Other versions

If you use the software, please consider citing scikit-learn.

3.2.4.3.1. `sklearn.ensemble.RandomForestClassifier`

3.2.4.3.1.1. Examples using `sklearn.ensemble.RandomForestClassifier`

3.2.4.3.1. `sklearn.ensemble`.RandomForestClassifier

class `sklearn.ensemble.`**`RandomForestClassifier`**(*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None*) ¶ [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

Parameters:

n_estimators : integer, optional (default=10)

The number of trees in the forest.

criterion : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

max_features : int, float, string or None, optional (default="auto")

Random Forest

Import the Random Forest function

```
from sklearn.ensemble import RandomForestClassifier
```

Create an instance of the model

```
model = RandomForestClassifier(random_state=0)
```

Random Forest

Run the cross-validation function using the Random Forest algorithm

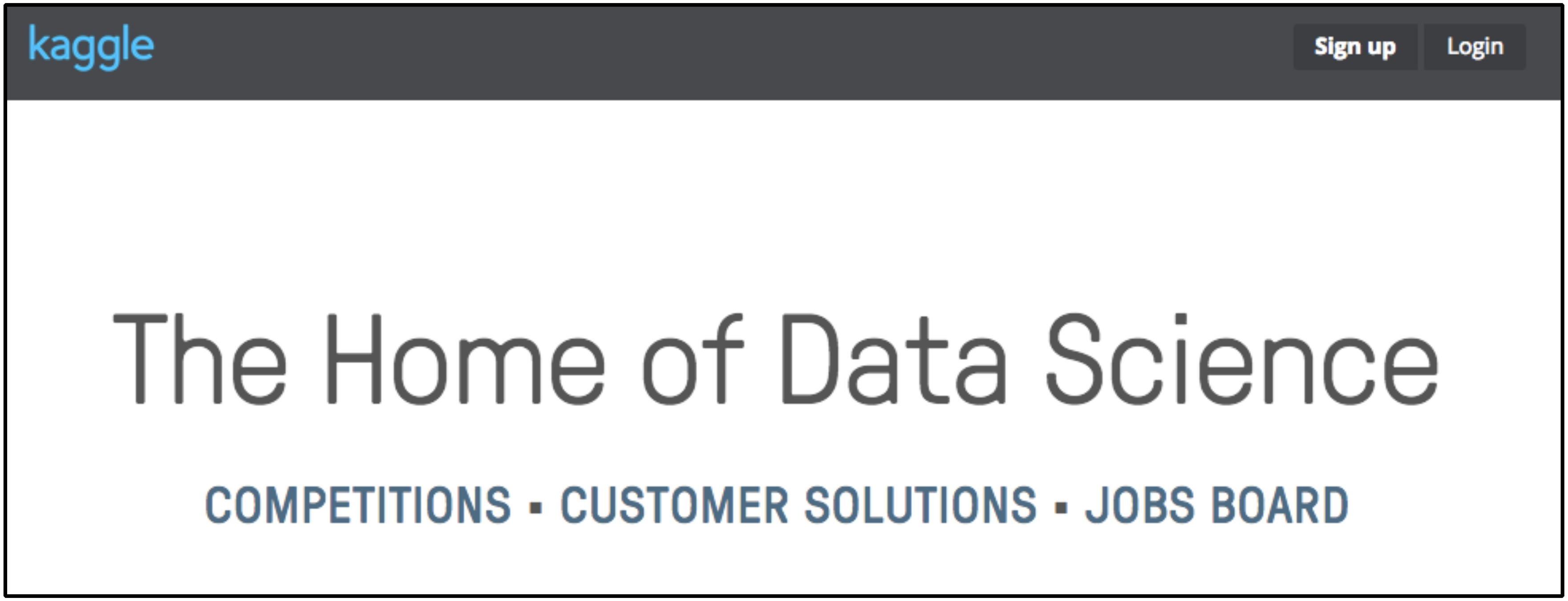
```
cross_validate(features, target, model, 10, 0)
```

Investigate feature importances

```
print model.fit(features, target).feature_importances_
```

Kaggle

- Data prediction competitions



Kaggle

- Create an account
- Find “Titanic: Machine Learning from Disaster”
- Submission Instructions:

  “You should submit a csv file with exactly 418 entries plus a header row. This must have exactly 2 columns: PassengerId (which can be sorted in any order), and Survived which contains your binary predictions: 1 for survived, 0 for did not.”

Make Predictions

- Train your machine learning algorithm with training data
- Read the test.csv file
- Clean test data
- Predict outcomes for test data
- Convert predictions into a DataFrame
- Save DataFrame as a CSV file (remember to exclude the index)
- Submit predictions to Kaggle site

Train Model on All Data

Train the model using the best data available

```
model = RandomForestClassifier(
        random_state=0).fit(features,target)
```

Read test.csv Data

Make sure you investigate the data

```
test_data = pd.read_csv('test.csv')
print test_data.head()
print test_data.info()
print test_data.describe()
```

Clean and Prep the Data

Clean text and missing values

```
test_data.Sex = test_data.Sex.replace(
        ['male','female'],[True,False])
avg_age = data.Age.mean()
test_data.Age = test_data.Age.fillna(avg_age)
```

Convert Pclass to dummies and merge to data

```
pclass = pd.get_dummies(test_data.Pclass,
                        prefix = 'Pclass')
test_data = pd.merge(test_data, pclass,
                left_index=True, right_index=True)
```

Clean and Prep the Data

Select features from test data and convert to numpy array

```
test_features = test_data[[‘Age', ‘Sex', ‘Pclass_1',
            ‘Pclass_2', 'Pclass_3']].values
```

Make Predictions

Create Predictions

```
predictions =  model.predict(test_features)
```

Add Predictions as new column in DataFrame

```
test_data['Survived'] = predictions
```

Save as CSV (make sure you set index=False)

```
kaggle = test_data[['PassengerId','Survived']]
kaggle.to_csv('kaggle_titanic_submission.csv',
                    index=False)
```

What We've Accomplished

- Built foundation of Python skills
- Acquired and stored data in variety of formats (e.g., csv, Excel, SQL, JSON)
- Used API to request data
- Explored, cleaned and summarized data
- Created data visualizations
- Implemented machine learning algorithms

Coding Best Practices

PEP-8 Style Guide

- https://www.python.org/dev/peps/pep-0008/
- maximum line length of 79 characters
- indentation
- line continuation
- commenting

Hard to Use at First But Review Once a Month

Next Steps

- Practice, Practice, Practice
  - Find a fun project
  - Lots of public data sets and web data to scrape
  - Pair programming buddy
- Join a Python Meetup
  - San Francisco Python - recommend Project Night event
  - Bay Area Python Interest Group (BayPIGgies)

Next Steps

- Keep Learning
  - Pycon and Pydata videos on YouTube
  - Python for Data Analysis (O'Reilly)
  - Machine Learning Classes (Coursera, Edx)
  - datasciencemasters.org
  - Kaggle competitions
- Don't Be Afraid to Ask for Help
- Keep in Touch

yelp.com/biz/quantsprout-san-francisco