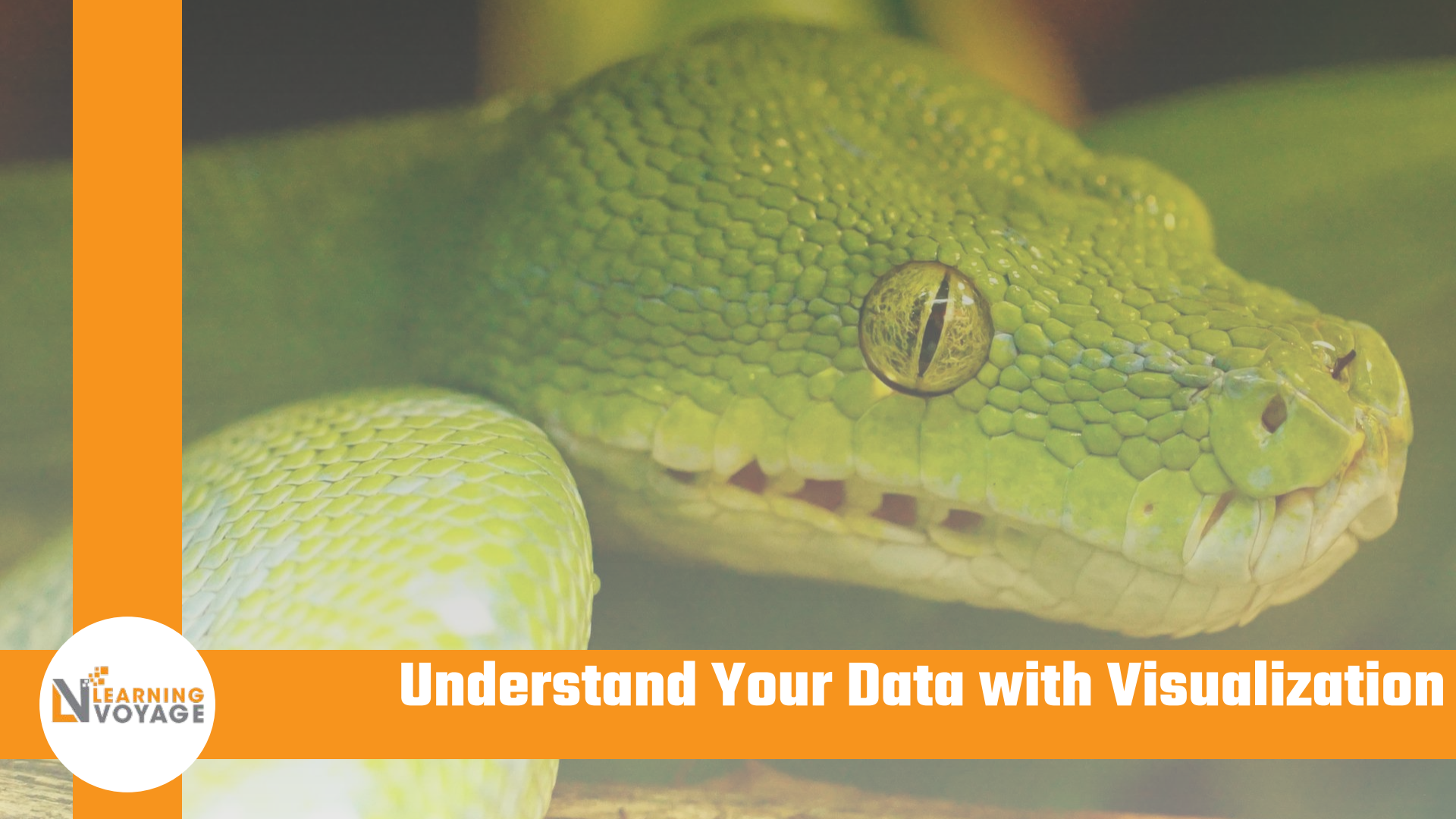


Understand Your Data with Visualization

Histograms


```
# Univariate Histograms from matplotlib import pyplot from
pandas import read_csv filename
='pima-indians-diabetes.data.csv' names =
['preg','plas','pres','skin','test','mass','pedi','age','c
lass'] data = read_csv(filename, names=names) data.hist()
pyplot.show()
```

Density plots

```
# Univariate Density Plots from matplotlib import pyplot
from pandas import read_csv filename
='pima-indians-diabetes.data.csv' names =
['preg','plas','pres','skin','test','mass','pedi','age','c
lass'] data = read_csv(filename, names=names)
data.plot(kind='density', subplots=True, layout=(3,3),
sharex=False) pyplot.show()
```

Box and Whisker Plots

```
# Box and Whisker Plots from matplotlib import pyplot from
pandas import read_csv filename =
"pima-indians-diabetes.data.csv" names =
['preg','plas','pres','skin','test','mass','pedi','age','c
lass'] data = read_csv(filename, names=names)
data.plot(kind='box', subplots=True, layout=(3,3),
sharex=False, sharey=False) pyplot.show()
```

Multivariate Plots

Correlation Matrix Plot

```
# Correlation Matrix Plot from matplotlib import pyplot from
pandas import read_csv import numpy filename
='pima-indians-diabetes.data.csv' names =
['preg','plas','pres','skin','test','mass','pedi','age','class
'] data = read_csv(filename, names=names) correlations =
data.corr() # plot correlation matrix fig = pyplot.figure() ax
= fig.add_subplot(111) cax = ax.matshow(correlations, vmin=-1,
vmax=1) fig.colorbar(cax) ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks) ax.set_yticks(ticks)
ax.set_xticklabels(names) ax.set_yticklabels(names)
pyplot.show()
```

```
# Correlation Matrix Plot (generic) from matplotlib import
pyplot from pandas import read_csv filename
='pima-indians-diabetes.data.csv' names =
['preg','plas','pres','skin','test','mass','pedi','age','c
lass'] data = read_csv(filename, names=names) correlations
= data.corr() # plot correlation matrix fig =
pyplot.figure() ax = fig.add_subplot(111) cax =
ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax) pyplot.show()
```

Scatter plot matrix

```
# Scatterplot Matrix from matplotlib import pyplot from
pandas import read_csv from pandas.plotting import
scatter_matrix filename = "pima-indians-diabetes.data.csv"
names =
['preg','plas','pres','skin','test','mass','pedi','age','c
lass'] data = read_csv(filename, names=names)
scatter_matrix(data) pyplot.show()
```

Scale Machine Learning Data

What we will cover

- . How to normalize your data from scratch.
- . How to standardize your data from scratch.
- . When to normalize as opposed to standardize data.

Why?

- Many machine learning algorithms expect the scale of the input and even the output data to be equivalent.
- It can help in methods that weight inputs in order to make a prediction, such as in linear regression and logistic regression.
- It is practically required in methods that combine weighted inputs in complex ways such as in artificial neural networks and deep learning.

Dataset

- Pima Indians Dataset

1. Normalize Data.
2. Standardize Data.
3. When to Normalize and Standardize.

Normalizing Data

- Normalization can refer to different techniques depending on context.
- Here, I am using normalization to refer to rescaling an input variable to the range between 0 and 1.
- Normalization requires that you know the minimum and maximum values for each attribute.

```
# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax
```

Test with a small dataset

x1 x2

50 30

20 90

```
# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax
```

- `# Contrive small dataset`
- `dataset = [[50, 30], [20, 90]]`
- `print(dataset)`
- `# Calculate min and max for each column`
- `minmax = dataset_minmax(dataset)`
- `print(minmax)`

Verify

[[50, 30], [20, 90]]

[[20, 50], [30, 90]]

- Once we have estimates of the maximum and minimum allowed values for each column, we can now normalize the raw data to the range 0 and 1.
- The calculation to normalize a single value for a column is:

$$\text{scaled value} = \frac{value - min}{max - min}$$

```
# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] -
            minmax[i][0])
```

```
# Contrive small dataset
dataset = [[50, 30], [20, 90]]
print(dataset)
# Calculate min and max for each column
minmax = dataset_minmax(dataset)
print(minmax)
# Normalize columns
normalize_dataset(dataset, minmax)
print(dataset)
```

Validation

- Running this example prints the output below, including the normalized dataset.

[[50, 30], [20, 90]]

[[20, 50], [30, 90]]

[[1, 0], [0, 1]]

normalize_diabetes.py

- We can combine this code with code for loading a CSV dataset and load and normalize the Pima Indians Diabetes dataset.
- The example first loads the dataset and converts the values for each column from string to floating point values.
- The minimum and maximum values for each column are estimated from the dataset, and finally, the values in the dataset are normalized.

- Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns
- BEFORE NORMALIZATION

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

- AFTER NORMALIZATION

[0.35294117647058826, 0.7437185929648241, 0.5901639344262295, 0.35353535353535354, 0.0,

0.5007451564828614, 0.2344150288898377, 0.4833333333333334, 1.0]

Standardize Data

- Standardization is a rescaling technique that refers to centering the distribution of the data on the value 0 and the standard deviation to the value 1.
- Together, the mean and the standard deviation can be used to summarize a normal distribution, also called the Gaussian distribution or bell curve.

$$\text{mean} = \frac{\sum_{i=1}^{n} values_i}{count(values)}$$

Define the Mean

```
# calculate column means
def column_means(dataset):
    means = [0 for i in range(len(dataset[0]))]
    for i in range(len(dataset[0])):
        col_values = [row[i] for row in dataset]
        means[i] = sum(col_values) / float(len(dataset))
    return means
```

Standard Deviation

- The standard deviation describes the average spread of values from the mean. It can be calculated as the square root of the sum of the squared difference between each value and the mean and dividing by the number of values minus 1.

$$\text{standard deviation} = \sqrt{\frac{\sum_{i=1}^{n}(value_i - mean)^2}{count(values) - 1}}$$

```
# calculate column standard deviations
def column_stdevs(dataset, means):
    stdevs = [0 for i in range(len(dataset[0]))]
    for i in range(len(dataset[0])):
        variance = [pow(row[i]-means[i], 2) for row in dataset]
        stdevs[i] = sum(variance)
    stdevs = [sqrt(x/(float(len(dataset)-1))) for x in stdevs]
    return stdevs
```

Validate

x1 x2

50 30

20 90

30 50

statistics_contrived_dataset.py


```
[[50, 30], [20, 90], [30, 50]]
[33.333333333333336, 56.666666666666664]
[15.275252316519467, 30.550504633038933]
```

That leads us to here...

- Once the summary statistics are calculated, we can easily standardize the values in each column.
- The calculation to standardize a given value is as follows:

$$\text{standardized\_value}_i = \frac{value_i - mean}{stdev}$$

standardize_contrived_dataset.py

```
# standardize dataset
def standardize_dataset(dataset, means, stdevs):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - means[i]) / stdevs[i]
```

Validate

[[50, 30], [20, 90], [30, 50]]

[33.333333333333336, 56.666666666666664]

[15.275252316519467, 30.550504633038933]

[[1.0910894511799618, -0.8728715609439694],
[-0.8728715609439697, 1.0910894511799962],

[-0.2182178902359253, -0.2182178902359923]]

standardize_diabetes.py

- Loaded data file pima-indians-diabetes.csv with 768 rows and 9 columns

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

[0.6395304921176576, 0.8477713205896718,
0.14954329852954296, 0.906679062347250 5,

-0.692439324724129, 0.2038799072674717, 0.468186870229798,
1.4250667195933604,

1.3650063669598067]

When to Standardize and When to Normalize

- Standardization is a scaling technique that assumes your data conforms to a normal distribution.
- If a given data attribute is normal or close to normal, this is probably the scaling method to use.
- It is good practice to record the summary statistics used in the standardization process so that you can apply them when standardizing data in the future that you may want to use with your model.
- Normalization is a scaling technique that does not assume any specific distribution.

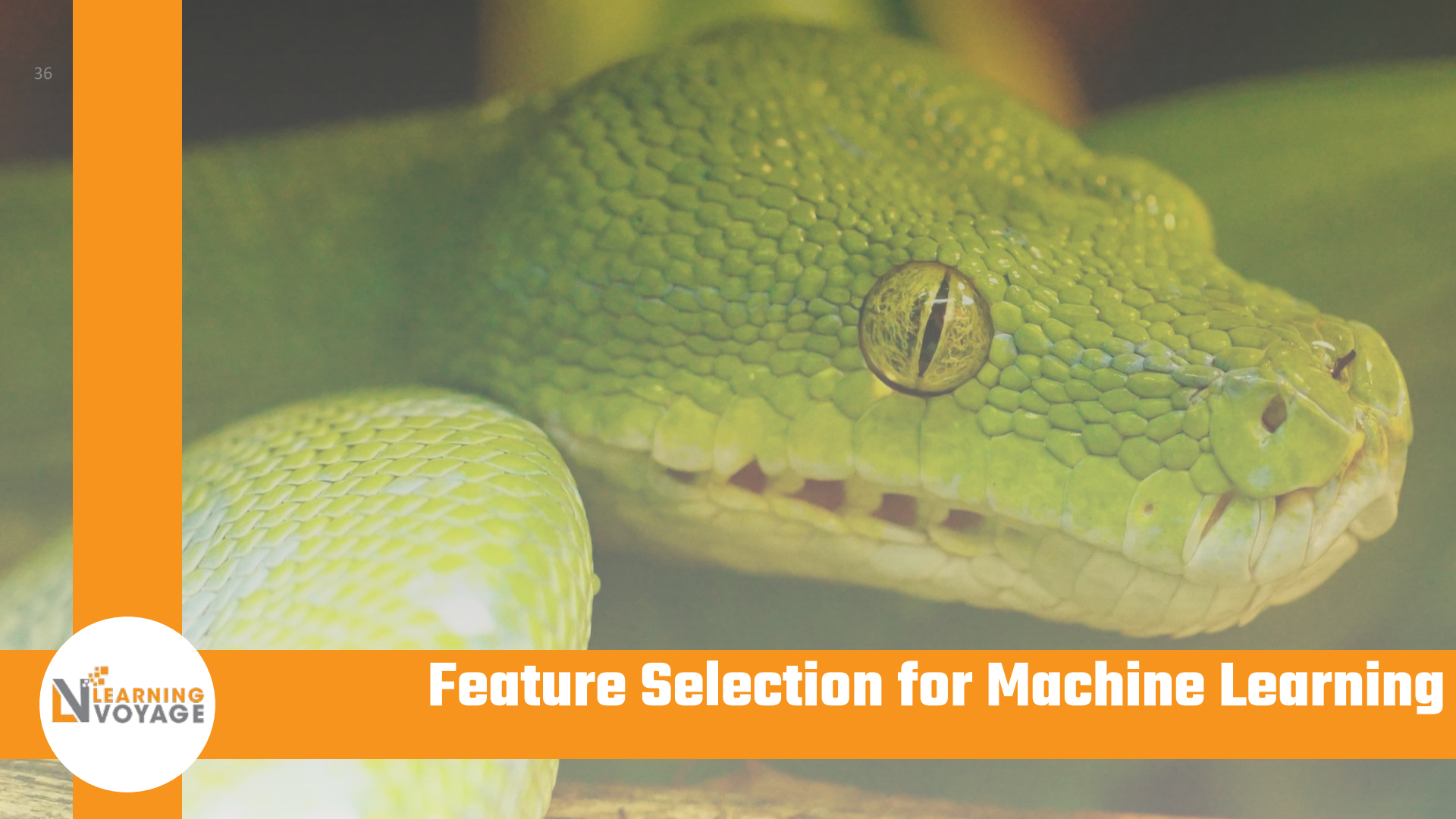
Other Things

- . Normalization that permits a congurable range, such as -1 to 1 and more.
- . Standardization that permits a congurable spread, such as 1, 2 or more standard deviations from the mean.
- . Exponential transforms such as logarithm, square root and exponents.
- . Power transforms such as Box-Cox for xing the skew in normally distributed data.

Summary

- . How to normalize data from scratch.
- . How to standardize data from scratch.
- . When to use normalization or standardization on your data.

- . Chapter 3 Data Pre-processing, page 27, Applied Predictive Modeling, 2013
- http://amzn.to/2e3lNXF



Feature Selection for Machine Learning

Feature Selection

- Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
- Improves Accuracy: Less misleading data means modeling accuracy improves.
- Reduces Training Time: Less data means that algorithms train faster.

Univariate Selection

```
# Feature Extraction-Univariate Statistical Test-Chi-squared for classification
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# load data
filename ='pima-indians-diabetes.data.csv'
names = ['preg','plas','pres','skin','test','mass','pedi','age','class'] dataframe =
read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

Univariate Selection

```
[ 111.52 1411.887 17.605 53.108 2175.565 127.669 5.393
  181.304]
[[ 148.  0.  33.6 50. ]
[ 85.  0.  26.6 31. ]
[ 183.  0.  23.3 32. ]
[ 89.  94.  28.1 21. ]
[ 137.  168.  43.1 33. ]]
```

Recursive Feature Selection

```
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
filename ='pima-indians-diabetes.data.csv'
names = ['preg','plas','pres','skin','test','mass','pedi','age','class'] dataframe =
read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression(solver= 'liblinear')
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

Recursive Feature Selection

```
Num Features: 3
Selected Features: [ True False False False False True True False]
Feature Ranking: [1 2 3 5 6 1 1 4]
```

Principal Component Analysis

```
# Feature Extraction with PCA
from pandas import read_csv
from sklearn.decomposition import PCA
# load data filename ='pima-indians-diabetes.data.csv'
names = ['preg','plas','pres','skin','test','mass','pedi','age','class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

Principal Component Analysis

```
Explained Variance: [ 0.88854663 0.06159078 0.02579012]

[[ -2.02176587e-03 9.78115765e-02 1.60930503e-02 6.07566861e-02
   9.93110844e-01 1.40108085e-02 5.37167919e-04 -3.56474430e-03] [
    2.26488861e-02 9.72210040e-01 1.41909330e-01 -5.78614699e-02
   -9.46266913e-02 4.69729766e-02 8.16804621e-04 1.40168181e-01] [
   -2.24649003e-02 1.43428710e-01 -9.22467192e-01 -3.07013055e-01
  2.09773019e-02 -1.32444542e-01 -6.39983017e-04 -1.25454310e-01]]
```

Feature Importance

```
# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
filename ='pima-indians-diabetes.data.csv'
names = ['preg','plas','pres','skin','test','mass','pedi','age','class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier(n_estimators=100)
model.fit(X, Y)
print(model.feature_importances_)
```

Feature Importance


```
[0.11029924 0.23133353 0.10086303 0.077177 0.07480096 0.14086308
 0.1162922 0.14837095]
```

Evaluate the Performance of Machine Learning Algorithms

Evaluate Machine Learning Algorithms

- Why can't you prepare your machine learning algortithm on your training set and use those predictions to evaluate performance?
  - The answer is simple: overfitting

Evaluate Machine Learning Algorithms

- ˆ Train and Test Sets.
- ˆ k-fold Cross-Validation.
- ˆ Leave One Out Cross-Validation.
- ˆ Repeated Random Test-Train Splits.

Split into Train and Test Sets

- The simplest method that we can use to evaluate the performance of a machine learning algorithm is to use different training and testing datasets.
- We can take our original dataset and split it into two parts.
- Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results.

Split into Train and Test Sets

- This algorithm evaluation technique is very fast.
- It is ideal for large datasets (millions of records) where there is strong evidence that both splits of the data are representative of the underlying problem.
- Because of the speed, it is useful to use this approach when the algorithm you are investigating is slow to train.
- A downside of this technique is that it can have a high variance.

```
#   Evaluate using a train and a test set from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8] test_size = 0.33 seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='liblinear') model.fit(X_train, Y_train)
result = model.score(X_test, Y_test) print("Accuracy: %.3f%%" % (result*100.0))
```

Accuracy: 75.591%

K-fold Cross-Validation

- Cross-validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split.
- It works by splitting the dataset into k-parts (e.g. k = 5 or k = 10).
- ***Each split of the data is called a fold.***
- The algorithm is trained on k- 1 folds with one held back and tested on the held back fold.

K-fold Cross-Validation

- For modest sized datasets in the thousands or tens of thousands of records, k values of 3, 5 and 10 are common.
- In the example we use 10-fold cross-validation.

```
#  Evaluate using Cross Validation
from pandas import read_csv
from sklearn.model_selection import KFold

from sklearn.model_selection import cross_val_score from sklearn.linear_model
import LogisticRegression filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename,
names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]

kfold = KFold(n_splits=10, random_state=7) model =
LogisticRegression(solver='liblinear') results = cross_val_score(model, X, Y,
cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Accuracy: 76.951% (4.841%)

Leave One Out Cross-Validation

- You can configure cross-validation so that the size of the fold is 1 (k is set to the number of observations in your dataset).
- This variation of cross-validation is called leave-one-out cross-validation.
- The result is a large number of performance measures that can be summarized in an effort to give a more reasonable estimate of the accuracy of your model on unseen data
- A downside is that it can be a computationally more expensive procedure than k-fold cross-validation.

```
# Evaluate using Leave One Out Cross Validation
from pandas import read_csv
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression filename =
'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename,
names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]
loocv = LeaveOneOut()
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=loocv)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Accuracy: 76.823% (42.196%)

Repeated Random Test-Train Splits

- Another variation on k-fold cross-validation is to create a random split of the data like the train/test split described above, but repeat the process of splitting and evaluation of the algorithm multiple times, like cross-validation.
- This has the speed of using a train/test split and the reduction in variance in the estimated performance of k-fold cross-validation.

```
# Evaluate using Shuffle Split Cross Validation
from pandas import read_csv
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression filename =
'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename,
names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8] n_splits = 10 test_size = 0.33 seed = 7
kfold = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Accuracy: 76.496% (1.698%)

What Techniques to Use When

- Generally ***k-fold cross-validation is the gold standard*** for evaluating the performance of a machine learning algorithm on unseen data with k set to 3, 5, or 10.
- Using a train/test split is good for speed when using a slow algorithm and produces performance estimates with lower bias when using large datasets.
- Techniques like leave-one-out cross-validation and repeated random splits can be useful intermediates when trying to balance variance in the estimated performance, model training speed and dataset size.

Summary

- Train and Test Sets.
- Cross-Validation.
- Leave One Out Cross-Validation.
- Repeated Random Test-Train Splits.

LEARNING VOYAGE

Evaluation Metrics

10

Machine Learning Algorithm Performance Metrics

- The metrics that you choose to evaluate your machine learning algorithms are very important.
- Choice of metrics influences how the performance of machine learning algorithms is measured and compared.
- They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.
- In this lecture, you will discover how to select and use different machine learning performance metrics in Python with scikit-learn.

Algorithm Evaluation Metrics

- For classification metrics, the Pima Indians onset of diabetes dataset is used as demonstration.
  - This is a binary classification problem where all of the input variables are numeric.

- For regression metrics, the Boston House Price dataset is used as demonstration.
  - This is a regression problem where all of the input variables are also numeric.

Algorithm Evaluation Metrics

- All techniques evaluate the same algorithms, Logistic Regression for Classification and Linear Regression for the regression problems.
- A 10-fold cross-validation test harness is used to demonstrate each metric, because this is the most likely scenario you will use when employing different algorithm evaluation metrics.

Classification Metrics

- ˆClassification Accuracy.
- ˆLogarithmic Loss.
- ˆArea Under ROC Curve.
- ˆConfusion Matrix.
- ˆClassification Report.

Classification Accuracy

- Classification accuracy is the number of correct predictions made as a ratio of all predictions made.
- This is the most common evaluation metric for Classification problems, it is also the most misused.
- It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case.

```
#  Cross Validation Classification Accuracy
from pandas import read_csv
from sklearn.model_selection import KFold

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression filename =
'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename,
names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]

kfold = KFold(n_splits=10, random_state=7) model =
LogisticRegression(solver='liblinear') scoring = 'accuracy'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring) print("Accuracy: %.3f (%.3f)" % (results.mean(),
results.std()))
```

Accuracy: 0.770 (0.048)

Logarithmic Loss

- Logarithmic loss (or logloss) is a performance metric for evaluating the predictions of probabilities of membership to a given class.
- The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm.
- Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction.
- Next is an example of calculating logloss for Logistic regression predictions on the Pima Indians onset of diabetes dataset.

```
#  Cross Validation Classification LogLoss
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score from sklearn.linear_model import LogisticRegression filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7) model = LogisticRegression(solver='liblinear') scoring
= 'neg_log_loss'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring) print("Logloss: %.3f (%.3f)" % (results.mean(),
results.std()))
```

Logloss: -0.493 (0.047)

Area Under ROC Curve

- Area under ROC Curve (or AUC for short) is a performance metric for binary Classification problems.
- The AUC represents a model's ability to discriminate between positive and negative classes.
  - An area of 1.0 represents a model that made all predictions perfectly.
  - An area of 0.5 represents a model that is as good as random.
- ROC can be broken down into sensitivity and specificity.
- A binary Classification problem is really a trade-o between sensitivity and specificity.

```
#  Cross Validation Classification ROC AUC from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score from sklearn.linear_model import LogisticRegression filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7) model = LogisticRegression(solver='liblinear') scoring
= 'roc_auc'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring) print("AUC: %.3f (%.3f)" % (results.mean(),
results.std()))
```

AUC: 0.824 (0.041)

Confusion Matrix

- The confusion matrix is a handy presentation of the accuracy of a model with two or more classes.
- The table presents predictions on the x-axis and true outcomes on the y-axis.
- The cells of the table are the number of predictions made by a machine learning algorithm.
- For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1.

```
#  Cross Validation Classification Confusion Matrix
from pandas import read_csv
from sklearn.model_selection import train_test_split from sklearn.linear_model import
LogisticRegression from sklearn.metrics import confusion_matrix filename =
'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8] test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='liblinear')
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[141  21]
 [ 41  51]]
```

Classification Report

- The scikit-learn library provides a convenience report when working on Classification problems to give you a quick idea of the accuracy of a model using a number of measures.
- The classification report() function displays the precision, recall, F1-score and support for each class.
- The example given next demonstrates the report on the binary Classification problem.

```
#  Cross Validation Classification Report
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8] test_size = 0.33 seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression(solver='liblinear') model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted) print(report)
```

Regression Metrics

- Mean Absolute Error.
- Mean Squared Error.
- R2.

Mean Absolute Error

- The Mean Absolute Error (or MAE) is the sum of the absolute differences between predictions and actual values.
- It gives an idea of how wrong the predictions were.
- The measure gives an idea of the magnitude of the error, but no idea of the direction (e.g. over or under predicting).

```
#  Cross Validation Regression MAE from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score from sklearn.linear_model import
LinearRegression filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

dataframe = read_csv(filename, delim_whitespace=True, names=names) array = dataframe.values
X = array[:,0:13] Y = array[:,13]
kfold = KFold(n_splits=10, random_state=7) model = LinearRegression()
scoring = 'neg_mean_absolute_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring) print("MAE: %.3f (%.3f)" % (results.mean(), results.std()))
```

Mean Squared Error

- The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error.
- Taking the square root of the mean squared error converts the units back to the original units of the output variable and can be meaningful for description and presentation.
- This is called the Root Mean Squared Error (or RMSE).

```
#  Cross Validation Regression MSE from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

dataframe = read_csv(filename, delim_whitespace=True, names=names) array = dataframe.values
X = array[:,0:13] Y = array[:,13]
kfold = KFold(n_splits=10, random_state=7) model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring) print("MSE: %.3f (%.3f)" % (results.mean(), results.std()))
```

R^2 Metric

- The R2 (or R Squared) metric provides an indication of the goodness of t of a set of predictions to the actual values.
- In statistical literature this measure is called the coefficient of determination.
- This is a value between 0 and 1 for no- t and perfect t respectively.

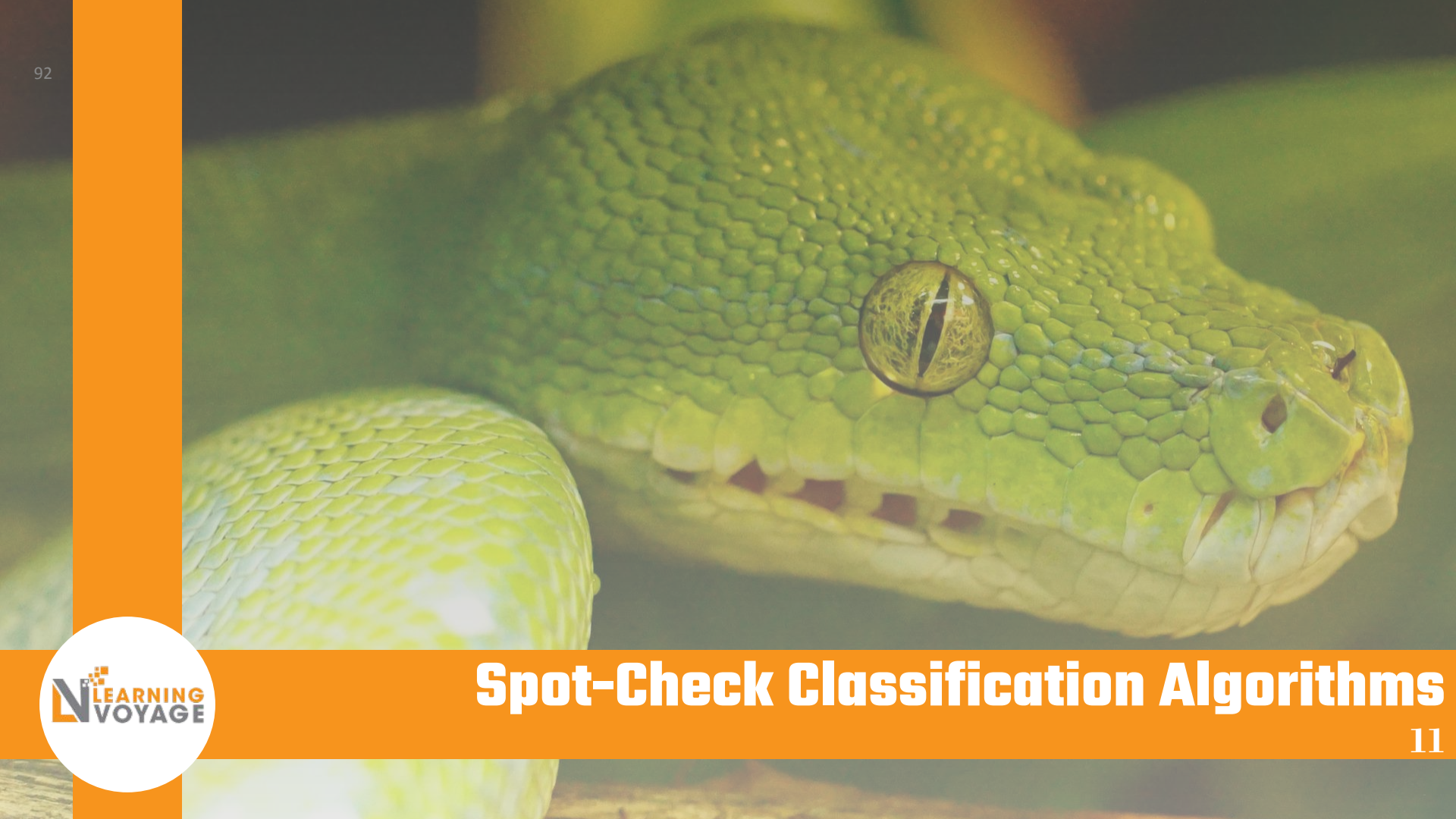
```
# Cross Validation Regression R^2 from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score from sklearn.linear_model import LinearRegression filename = housing.csv
names = [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B,LSTAT,MEDV]

dataframe = read_csv(filename, delim_whitespace=True, names=names) array = dataframe.values
X = array[:,0:13] Y = array[:,13]
kfold = KFold(n_splits=10, random_state=7)
model = LinearRegression()
scoring = r2
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("R^2: %.3f (%.3f)" % (results.mean(), results.std()))
```

Summary

In this section you discovered metrics that you can use to evaluate your machine learning algorithms.

You learned about three Classification metrics:

- Accuracy,
- Logarithmic Loss
- and Area Under ROC Curve.



LEARNING VOYAGE

Spot-Check Classification Algorithms

11

- . How to implement the random prediction algorithm.
- . How to implement the zero rule prediction algorithm.

- . Random Prediction Algorithm.
- . Zero Rule Algorithm.

Random Prediction Algo

- The random prediction algorithm predicts a random outcome as observed in the training data.
- It is perhaps the simplest algorithm to implement. It requires that you store all of the distinct outcome values in the training data, which could be large on regression problems with lots of distinct values.

```
# Generate random predictions
def random_algorithm(train, test):
    output_values = [row[-1] for row in train]
    unique = list(set(output_values))
    predicted = list()
    for row in test:
        index = randrange(len(unique))
        predicted.append(unique[index])
    return predicted
```

```
seed(1)
train = [[0], [1], [0], [1], [0], [1]]
test = [[None], [None], [None], [None]]
predictions = random_algorithm(train, test)
print(predictions)
```

Zero Rule Algorithm

- The Zero Rule Algorithm is a better baseline than the random algorithm.
- It uses more information about a given problem to create one rule in order to make predictions.
- This rule is different depending on the problem type.
- Let's start with classification problems, predicting a class label.

Classification

```
# zero rule algorithm for classification
def zero_rule_algorithm_classification(train, test):
    output_values = [row[-1] for row in train]
    prediction = max(set(output_values),
    key=output_values.count)
    predicted = [prediction for i in range(len(test))]
    return predicted
```

```
# Example of Zero Rule Classification Predictions
from random import seed
# zero rule algorithm for classification
def zero_rule_algorithm_classification(train, test):
    output_values = [row[-1] for row in train]
    prediction = max(set(output_values), key=output_values.count)
    predicted = [prediction for i in range(len(test))]
    return predicted
seed(1)
train = [['0'], ['0'], ['0'], ['0'], ['1'], ['1']]
test = [[None], [None], [None], [None]]
predictions =
```

Regression

- Regression problems require the prediction of a real value. A good default prediction for real values is to predict the central tendency.
- This could be the mean or the median.
- A good default is to use the mean (also called the average) of the output value observed in the training data.
- This is likely to have a lower error than random prediction which will return any observed output value.

$$mean = \frac{\sum_{i=1}^{n} value_i}{count(values)}$$

```
# zero rule algorithm for regression
def zero_rule_algorithm_regression(train, test):
    output_values = [row[-1] for row in train]
    prediction = sum(output_values) / float(len(output_values))
    predicted = [prediction for i in range(len(test))]
    return predicted
```

```
# Example of Zero Rule Regression Predictions
from random import seed
# zero rule algorithm for regression
def zero_rule_algorithm_regression(train, test):
    output_values = [row[-1] for row in train]
    prediction = sum(output_values) / float(len(output_values))
    predicted = [prediction for i in range(len(test))]
    return predicted
seed(1)
train = [[10], [15], [12], [15], [18], [20]]
test = [[None], [None], [None], [None]]
predictions = zero_rule_algorithm_regression(train, test)
print(predictions)
```

Other Things

- . Alternate Central Tendency where the median, mode or other central tendency calculations are predicted instead of the mean.
- . Moving Average for time series problems where the mean of the last n records is predicted.

Spot-Check Classication Algorithms

- You cannot know which algorithm will work best on your dataset beforehand.
- You must use trial and error to discover a shortlist of algorithms that do well on your problem that you can then double down on and tune further.
- I call this process spot-checking.

- Try a mixture of algorithm representations (e.g. instances and trees).
- Try a mixture of learning algorithms (e.g. different algorithms for learning the same type of representation).
- Try a mixture of modeling types (e.g. linear and nonlinear functions or parametric and nonparametric).

Algorithms Overview

- ˆLogistic Regression.
- ˆLinear Discriminant Analysis.
- Then looking at four nonlinear machine learning algorithms:
  - ˆ k-Nearest Neighbors.
  - ˆ Naive Bayes.
  - ˆ Classification and Regression Trees.
  - ˆ Support Vector Machines.

Linear Machine Learning Algorithms

This section demonstrates minimal recipes for how to use two linear machine learning algorithms:

1. logistic regression
2. and linear discriminant analysis.

Logistic Regression

- Logistic regression assumes a Gaussian distribution for the numeric input variables and can model binary Classification problems.
- You can construct a logistic regression model using the LogisticRegression class

```
#  Logistic Regression Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression(solver='liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Linear Discriminant Analysis

- Linear Discriminant Analysis or LDA is a statistical technique for binary and multiclass Classification.
- It too assumes a Gaussian distribution for the numerical input variables.
- You can construct an LDA model using the LinearDiscriminantAnalysis class

```
# LDA Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis filename = pima-indians-diabetes.data.csv
names = [preg, plas, pres, skin, test, mass, pedi, age, class] dataframe = read_csv(filename, names=names)
array = dataframe.values

X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = LinearDiscriminantAnalysis()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Nonlinear Machine Learning Algorithms

- This section demonstrates minimal recipes for how to use 4 nonlinear machine learning algorithms.

k-Nearest Neighbors

- The k-Nearest Neighbors algorithm (or KNN) uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction.
- You can construct a KNN model using the KNeighborsClassifier class3

```
# KNN Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename, names=names) array = dataframe.values

X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = KNeighborsClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Naive Bayes

- Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value.
- These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption).
- When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for input variables using the Gaussian Probability Density Function.
- You can construct a Naive Bayes model using the GaussianNB class

```
#  Gaussian Naive Bayes Classification from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe = read_csv(filename, names=names)
array = dataframe.values X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7) model = GaussianNB()
results = cross_val_score(model, X, Y, cv=kfold) print(results.mean())
```

Classification and Regression Trees

- Classification and Regression Trees (CART or just decision trees) construct a binary tree from the training data.
- Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index).
- You can construct a CART model using the DecisionTreeClassifier class

```
# CART Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values

X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = DecisionTreeClassifier()
results = cross_val_score(model, X, Y, cv=kfold)

print(results.mean())
```

Support Vector Machines

- Support Vector Machines (or SVM) seek a line that best separates two classes.
- Those data instances that are closest to the line that best separates the classes are called support vectors and in uence where the line is placed.
- SVM has been extended to support multiple classes. Of particular importance is the use of different kernel functions via the kernel parameter.
- A powerful Radial Basis Function is used by default.
- You can construct an SVM model using the SVC class

```
# SVM Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
filename = pima-indians-diabetes.data.csv
names = [preg, plas, pres, skin, test, mass, pedi, age, class] dataframe = read_csv(filename, names=names) array = dataframe.values

X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = SVC(gamma=auto)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Summary

- In this chapter you discovered 6 machine learning algorithms that you can use to spot-check on your Classification problem in Python using scikit-learn.
- Specifically, you learned how to spot-check two linear machine learning algorithms: Logistic Regression and Linear Discriminant Analysis.
- You also learned how to spot-check four nonlinear algorithms: k-Nearest Neighbors, Naive Bayes, Classification and Regression Trees and Support Vector Machines