

Introduction	
Creating a row Vector	<code>np.array([1, 2, 3])</code>
Creating a column Vector	<code>np.array([[1], [2], [3]])</code>
Creating a Matrix	<code>np.array([[1, 2], [1, 2], [1, 2]])</code>
Creating a Sparse Matrix	<code>from scipy import sparse</code> <code>sparse.csr_matrix(matrix)</code> #shows the indexes of non zero elements
Select all elements of a vector	<code>vector[:]</code>
Select all rows and the second column	<code>matrix[:,1:2]</code>
View number of rows and columns	<code>matrix.shape</code>
View number of elements	<code>matrix.size</code>
View number of dimensions	<code>matrix.ndim</code>
Applying Operations to Elements	<code>add_100 = lambda i: i + 100</code> <code>vectorized_add_100 = np.vectorize(add_100)</code> <code>vectorized_add_100(matrix)</code>
maximum value in an array	<code>np.max(matrix)</code>

Introduction (cont)	
minimum value in an array	<code>np.min(matrix)</code>
Return mean	<code>np.mean(matrix)</code>
Return variance	<code>np.var(matrix)</code>
Return standard deviation	<code>np.std(matrix)</code>
Reshaping Arrays	<code>matrix.reshape(2, 6)</code>
Transposing a Vector or Matrix	<code>matrix.T</code>
You need to transform a matrix into a one-dimensional array	<code>matrix.flatten()</code>
Return matrix rank (This corresponds to the maximal number of linearly independent columns of the matrix)	<code>np.linalg.matrix_rank(matrix)</code>
Calculating the Determinant	<code>np.linalg.det(matrix)</code>
Getting the Diagonal line of a Matrix	<code>matrix.diagonal(offset=1)</code> (offsets the diagonal by the amount we put, can be negative))
Return trace (sum of the diagonal elements)	<code>matrix.trace()</code>
Finding Eigenvalues and Eigenvectors	<code>eigenvalues, eigenvectors = np.linalg.eig(matrix)</code>

Introduction (cont)	
Calculating Dot Products (sum of the product of the elements of two vectors)	<code>np.dot(vector_a, vector_b)</code>
Add two matrices	<code>np.add(matrix_a, matrix_b)</code>
Subtract two matrices	<code>np.subtract(matrix_a, matrix_b)</code>
	Alternatively, we can simply use the + and - operators
Multiplying Matrices	<code>np.dot(matrix_a, matrix_b)</code>
	Alternatively, in Python 3.5+ we can use the @ operator
Multiply two matrices element-wise	<code>matrix_a * matrix_b</code>
Inverting a Matrix	<code>p.linalg.inv(matrix)</code>
Set seed for random value generation	<code>np.random.seed(0)</code>
Generate three random floats between 0.0 and 1.0	<code>np.random.random(3)</code>
Generate three random integers between 1 and 10	<code>np.random.randint(0, 11, 3)</code>
Draw three numbers from a normal distribution with mean 0.0 and standard deviation of 1.0	<code>np.random.normal(0.0, 1.0, 3)</code>



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 1 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Introduction (cont)

Draw three numbers from a logistic distribution with mean 0.0 and scale of 1.0

```
np.random.logistic(0.0, 1.0, 3)
```

Draw three numbers greater than or equal to 1.0 and less than 2.0

```
np.random.uniform(1.0, 2.0, 3)
```

We select element from matrixes and vectors like we do in R.

```
# Find maximum element in each column
np.max(matrix, axis=0) -> array([7, 8, 9])
```

One useful argument in reshape is -1, which effectively means "as many as needed," so reshape(1, -1) means one row and as many columns as needed:

Clustering

Clustering Using K-Means

Load libraries

```
from sklearn.cluster import KMeans
```

Create k-mean object

```
cluster = KMeans(n_clusters=3, random_state=0, n_jobs=-1)
```

Train model

```
model = cluster.fit(features_std)
```

Predict observation's cluster

```
model.predict(new_observation)
```

View predict class

```
model.labels_
```

Speeding Up K-Means Clustering

Load libraries

```
from sklearn.cluster import MiniBatchKMeans
```

Clustering (cont)

Create k-mean object

```
cluster = MiniBatchKMeans(n_clusters=3, random_state=0, batch_size=100)
```

Train model

```
model = cluster.fit(features_std)
```

Clustering Using MeanShift

group observations without assuming the number of clusters or their shape

Load libraries

```
from sklearn.cluster import MeanShift
```

Create meanshift object

```
cluster = MeanShift(n_jobs=-1)
```

Train model

```
model = cluster.fit(features_std)
```

Note on meanshift

cluster_all=False wherein orphan observations are given the label of -1

Clustering Using DBSCAN

group observations into clusters of high density

Load libraries

```
from sklearn.cluster import DBSCAN
```

Create meanshift object

```
cluster = DBSCAN(n_jobs=-1)
```

Train model

```
model = cluster.fit(features_std)
```

DBSCAN has three main parameters to set:

Clustering (cont)

eps

The maximum distance from an observation for another observation to be considered its neighbor.

min_samples

The minimum number of observations less than eps distance from an observation for it to be considered a core observation.

metric

The distance metric used by eps—for example, minkowski or euclidean

Clustering Using Hierarchical Merging

Load libraries

```
from sklearn.cluster import AgglomerativeClustering
```

Create meanshift object

```
cluster = AgglomerativeClustering(n_clusters=3)
```

Train model

```
model = cluster.fit(features_std)
```

AgglomerativeClustering uses the linkage parameter to determine the merging strategy to minimize the following:



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 2 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Clustering (cont)

Average distance between observations from pairs of clusters (average)

Maximum distance between observations from pairs of clusters (complete)

MiniBatchKMeans works similarly to KMeans, with one significant difference: the `batch_size` parameter. `batch_size` controls the number of randomly selected observations in each batch.

Handling Categorical Data

Encoding Nominal Categorical Features from `sklearn.preprocessing` import `LabelBinarizer`, `MultiLabelBinarizer`

Create one-hot encoder `one_hot = LabelBinarizer()`

One-hot encode feature `one_hot.fit_transform(feature)`

View feature classes `one_hot.classes_`

reverse the one-hot encoding `one_hot.inverse_transform(one_hot.transform(feature))`

Create dummy variables from feature `pd.get_dummies(feature[:,0])`

Create multiclass one-hot encoder `one_hot_multiclass = MultiLabelBinarizer()`

Handling Categorical Data (cont)

One-hot encode multiclass feature `one_hot_multiclass.fit_transform(multiclass_feature)`

see the classes with the `classes_` method `one_hot_multiclass.classes_`

Encoding Ordinal Categorical Features `dataframe["Score"].replace(dic with categoricals as keys and numbers as values)`

Encoding Dictionaries of Features from `sklearn.feature_extraction` import `DictVectorizer`

Create dictionary `data_dict = [{"Red": 2, "Blue": 4}, {"Red": 4, "Blue": 3}, {"Red": 1, "Yellow": 2}, {"Red": 2, "Yellow": 2}]`

Create dictionary vectorizer `dictvectorizer = DictVectorizer(sparse=False)`

Convert dictionary to feature matrix `features = dictvectorizer.fit_transform(data_dict)`

Get feature names `feature_names = dictvectorizer.get_feature_names()`

Imputing Missing Class Values from `sklearn.neighbors` import `KNeighborsClassifier`

Handling Categorical Data (cont)

Train KNN learner `clf = KNeighborsClassifier(3, weights='distance')`
`trained_model = clf.fit(X[:,1:], X[:,0])`

Predict missing values' class `imputed_values = trained_model.predict(X_with_nan[:,1:])`

Join column of predicted class with their other features `X_with_imputed = np.hstack((imputed_values.reshape(-1,1), X_with_nan[:,1:]))`

Join two feature matrices `np.vstack((X_with_imputed, X))`

Use imputer to fill most frequent value `imputer = Imputer(strategy='most_frequent', axis=0)`

Handling Imbalanced Classes `RandomForestClassifier(class_weight="balanced")`

downsample the majority class `i_class0 = np.where(-target == 0)[0]`

`i_class1 = np.where(-target == 1)[0]`

Number of observations in each class `n_class0 = len(i_class0)`

`n_class1 = len(i_class1)`



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 3 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Handling Categorical Data (cont)

For every observation of class 0, randomly sample from class 1 without replacement

```
i_class1_downsampled = np.random.choice(i_class1, size=n_class0, replace=False)
```

Join together class 0's target vector with the downsampled class 1's target vector

```
np.hstack((target[i_class0], target[i_class1_downsampled]))
```

Join together class 0's feature matrix with the downsampled class 1's feature matrix

```
np.vstack((features[i_class0,:], features[i_class1_downsampled,:]))[0:5]
```

upsample the minority class

```
i_class0_upsampled = np.random.choice(i_class0, size=n_class1, replace=True)
```

Join together class 0's upsampled target vector with class 1's target vector

```
np.concatenate((target[i_class0_upsampled], target[i_class1]))
```

Handling Categorical Data (cont)

Join together class 0's upsampled feature matrix with class 1's feature matrix

```
np.vstack((features[i_class0_upsampled,:], features[i_class1,:]))[0:5]
```

A second strategy is to use a model evaluation metric better suited to imbalanced classes. Accuracy is often used as a metric for evaluating the performance of a model, but when imbalanced classes are present accuracy can be ill suited. Some better metrics we discuss in later chapters are confusion matrices, precision, recall, F1 scores, and ROC curves

Dimensionality Reduction Using Feature Extraction

Reducing Features Using Principal Components

```
from sklearn.decomposition import PCA
```

```
from sklearn.preprocessing import StandardScaler
```

Standardize the feature matrix

```
features = StandardScaler().fit_transform(digits.data)
```

Create a PCA that will retain 99% of variance

```
pca = PCA(n_components=0.99, whiten=True)
```

Conduct PCA

```
features_pca = pca.fit_transform(features)
```

Dimensionality Reduction Using Feature Extraction (cont)

Reducing Features When Data Is Linearly Inseparable

Use an extension of principal component analysis that uses kernels to allow for non-linear dimensionality reduction from `sklearn.decomposition`

```
import PCA, KernelPCA
```

Apply kernel PCA with radius basis function (RBF) kernel

```
kpca = KernelPCA(kernel="rbf", gamma=15, n_components=1)
```

```
features_kpca = kpca.fit_transform(features)
```

Reducing Features by Maximizing Class Separability

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

Create and run an LDA, then use it to transform the features

```
LinearDiscriminantAnalysis(n_components=1)
```

```
features_lda = lda.fit(features, target).transform(features)
```

amount of variance explained by each component

```
lda.explained_variance_ratio_
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 4 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Dimensionality Reduction Using Feature Extraction (cont)

non-negative matrix factorization (NMF) to reduce the dimensionality of the feature matrix

Create, fit, and apply NMF

```
nmf = NMF(n_components=10,
random_state=1)
```

```
features_nmf =
nmf.fit_transform(-
features)
```

Reducing Features on Sparse Data (Truncated Singular Value Decomposition (TSVD))

```
from sklearn.deco-
position import
TruncatedSVD
```

```
from scipy.sparse
import csr_matrix
```

Standardize feature matrix

```
features = Stand-
ardScaler().fit_tra-
nsform(digits.data)
```

Make sparse matrix

```
features_sparse =
csr_matrix(featur-
es)
```

Create a TSVD

```
tsvd = Truncated-
SVD(n_compon-
ents=10)
```

Conduct TSVD on sparse matrix

```
features_sparse-
_tsvd = tsvd.fit(fea-
tures_sparse).t-
ransform(featur-
es_sparse)
```

Dimensionality Reduction Using Feature Extraction (cont)

Sum of first three components' explained variance ratios

```
tsvd.explained_-
variance_ratio_-
[0:3].sum()
```

196 e 200

One major requirement of NMF is that, as the name implies, the feature matrix cannot contain negative values.

Trees and Forests

Training a Decision Tree Classifier

```
from sklearn.tree import
DecisionTreeClassifier
```

Create decision tree classifier object

```
decisiontree = DecisionT-
reeClassifier(random_-
state=0)
```

Train model

```
model = decisiontree.fit(f-
eatures, target)
```

Predict observation's class

```
model.predict(observa-
tion)
```

Training a Decision Tree Regressor

```
from sklearn.tree import
DecisionTreeRegressor
```

Create decision tree classifier object

```
decisiontree = DecisionT-
reeRegressor(random_s-
tate=0)
```

Train model

```
model = decisiontree.fit(f-
eatures, target)
```

Create decision tree classifier object using entropy

```
decisiontree_mae =
DecisionTreeRegressor-
(criterion="mae",
random_state=0)
```

Trees and Forests (cont)

Visualizing a Decision Tree Model

```
from IPython.display import
Image
```

```
import pydotplus
```

```
from sklearn import tree
```

Create DOT data

```
dot_data = tree.export_graphv-
iz(decisiontree, out_file=None,
feature_names=iris.feature_-
names, class_names=iris.targ-
et_names)
```

Draw graph

```
graph = pydotplus.graph_from_-
dot_data(dot_data)
```

Show graph

```
Image(graph.create_png())
```

Create PDF

```
graph.write_pdf("iris.pdf")
```

Create PNG

```
graph.write_png("iris.png")
```

Training a Random Forest Classifier

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

Create random forest classifier object

```
randomforest = RandomForest-
Classifier(random_state=0,
n_jobs=-1)
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 5 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Trees and Forests (cont)

Create random forest classifier object using entropy

```
randomforest_entropy = RandomForestClassifier(criterion="entropy", random_state=0)
```

Training a Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
```

Create random forest classifier object

```
randomforest = RandomForestRegressor(random_state=0, n_jobs=-1)
```

Identifying Important Features in Random Forests

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

Create random forest classifier object

```
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1)
```

Calculate feature importances

```
importances = model.feature_importances_
```

Sort feature importances in descending order

```
indices = np.argsort(importances)[::-1]
```

Rearrange feature names so they match the sorted feature importances

```
names = [iris.feature_names[i] for i in indices]
```

Create plot

```
plt.figure()
```

Create plot title

```
plt.title("Feature Importance")
```

Trees and Forests (cont)

Add bars

```
plt.bar(range(features.shape[1]), importances[indices])
```

Add feature names as x-axis labels

```
plt.xticks(range(features.shape[1]), names, rotation=90)
```

Show plot

```
plt.show()
```

Selecting Important Features in Random Forests

```
from sklearn.feature_selection import SelectFromModel
```

Create random forest classifier

```
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1)
```

Create object that selects features with importance greater than or equal to a threshold

```
selector = SelectFromModel(randomforest, threshold=0.3)
```

Feature new feature matrix using selector

```
features_important = selector.fit_transform(features, target)
```

Train random forest using most important features

```
model = randomforest.fit(features_important, target)
```

Handling Imbalanced Classes

```
Train a decision tree or random forest model with class_weight="balanced"
```

Trees and Forests (cont)

Create random forest classifier object

```
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1, class_weight="balanced")
```

Controlling Tree Size

Create decision tree classifier object

```
decisiontree = DecisionTreeClassifier(random_state=0, max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0, max_leaf_nodes=None, min_impurity_decrease=0)
```

Improving Performance Through Boosting

```
from sklearn.ensemble import AdaBoostClassifier
```

Create adaboost tree classifier object

```
adaboost = AdaBoostClassifier(random_state=0)
```

Evaluating Random Forests with Out-of-Bag Errors

You need to evaluate a random forest model without using cross-validation

Create random tree classifier object

```
randomforest = RandomForestClassifier(random_state=0, n_estimators=1000, oob_score=True, n_jobs=-1)
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 6 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Trees and Forests (cont)

OOB scores of a random forest `oob_score_`

Trees and Forests

Training a Decision Tree Classifier

Load libraries `from sklearn.tree import DecisionTreeClassifier`

Create decision tree classifier object `decisiontree = DecisionTreeClassifier(random_state=0)`

Train model `model = decisiontree.fit(features, target)`

Training a Decision Tree Regressor

Use scikit-learn's DecisionTreeRegressor `from sklearn.tree import DecisionTreeRegressor`

Create decision tree classifier object `decisiontree = DecisionTreeRegressor(random_state=0)`

Train model `model = decisiontree.fit(features, target)`

Linear Regression

Fitting a Line

Load libraries `from sklearn.linear_model import LinearRegression`

Create linear regression `regression = LinearRegression()`

Linear Regression (cont)

Fit the linear regression `model = regression.fit(features, target)`

Handling Interactive Effects You have a feature whose effect on the target variable depends on another feature.

Load libraries `from sklearn.preprocessing import PolynomialFeatures`

Create interaction term `interaction = PolynomialFeatures(degree=3, include_bias=False, interaction_only=True)`

`features_interaction = interaction.fit_transform(features)`

Create linear regression `regression = LinearRegression()`

Fit the linear regression `model = regression.fit(features_interaction, target)`

Fitting a Nonlinear Relationship Create a polynomial regression by including polynomial features in a linear regression model

Load library `from sklearn.preprocessing import PolynomialFeatures`

Linear Regression (cont)

Create polynomial features x^2 and x^3 `polynomial = PolynomialFeatures(degree=3, include_bias=False)`

`features_polynomial = polynomial.fit_transform(features)`

Create linear regression `regression = LinearRegression()`

Fit the linear regression `model = regression.fit(features_polynomial, target)`

Reducing Variance with Regularization

Use a learning algorithm that includes a shrinkage penalty (also called regularization) like ridge regression and lasso regression:

Load libraries `from sklearn.linear_model import Ridge`

Create ridge regression with an alpha value `regression = Ridge(alpha=0.5)`

Fit the linear regression `model = regression.fit(features_standardized, target)`

Load library `from sklearn.linear_model import RidgeCV`

Create ridge regression with three alpha values `regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])`

Fit the linear regression `model_cv = regr_cv.fit(features_standardized, target)`



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 7 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Linear Regression (cont)

View coefficients	<code>model_cv.coef_</code>
View alpha	<code>model_cv.alpha_</code>
Reducing Features with Lasso Regression	You want to simplify your linear regression model by reducing the number of features.
Load library	<code>from sklearn.linear_model import Lasso</code>
Create lasso regression with alpha value	<code>regression = Lasso(alpha=0.5)</code>
Fit the linear regression	<code>model = regression.fit(features_standardized, target)</code>
Create lasso regression with a high alpha	<code>regression_a10 = Lasso(alpha=10)</code> <code>model_a10 = regression_a10.fit(features_standardized, target)</code>

`interaction_only=True` tells PolynomialFeatures to only return interaction terms
PolynomialFeatures will add a feature containing ones called a bias. We can prevent that with `include_bias=False`
Polynomial regression is an extension of linear regression to allow us to model nonlinear relationships.

Loading Data

Loading a Sample Dataset	<code>from sklearn import datasets</code> <code>digits = datasets.load_digits()</code> <code>features = digits.data</code> <code>target = digits.target</code>
Creating a Simulated Dataset for regression	<code>from sklearn.datasets import make_regression</code> <code>features, target, coefficients = make_regression(n_samples = 100, n_features = 3, n_informative = 3, n_targets = 1, noise = 0.0, coef = True, random_state = 1)</code>
Creating a Simulated Dataset for classification	<code>from sklearn.datasets import make_classification</code> <code>features, target = make_classification(n_samples = 100, n_features = 3, n_informative = 3, n_redundant = 0, n_classes = 2, weights = [.25, .75], random_state = 1)</code>

Loading Data (cont)

Creating a Simulated Dataset for clustering	<code>from sklearn.datasets import make_blobs</code> <code>features, target = make_blobs(n_samples = 100, n_features = 2, centers = 3, cluster_std = 0.5, shuffle = True, random_state = 1)</code>
Loading a CSV File	<code>dataframe = pd.read_csv(data, sep=',')</code>
Loading an Excel File	<code>pd.read_excel(url, sheetname=0, header=1)</code> If we need to load multiple sheets, include them as a list.
Loading a JSON File	<code>pd.read_json(url, orient='columns')</code> The key difference is the orient parameter, which indicates to pandas how the JSON file is structured. However, it might take some experimenting to figure out which argument (split, records, index, columns, and values) is the right one.



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 8 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Loading Data (cont)

```

convert semistruc-   json_normalize
tured JSON data
into a pandas
DataFrame

Querying a SQL      from sqlalchemy
Database            import create_engine

                    database_connection
                    = create_engine('sql-
                    ite:///sample.db')

                    pd.read_sql_que-
                    ry('SELECT * FROM
                    data', database_con-
                    nection)
    
```

In addition, `make_classification` contains a `weights` parameter that allows us to simulate datasets with imbalanced classes. For example, `weights = [.25,.75]`

For `make_blobs`, the `centers` parameter determines the number of clusters generated.

Naive Bayes

```

Training a Classifier for   Use a
Continuous Features       Gaussian naive
                          Bayes
                          classifier

Load libraries             from sklearn.n-
                          aive_bayes
                          import
                          GaussianNB

Create Gaussian Naive     classifier =
Bayes object              GaussianNB()

Train model                model = classi-
                          fier.fit(features,
                          target)

Create Gaussian Naive     clf = Gaussi-
Bayes object with prior   anNB(priors=-
probabilities of each    [0.25, 0.25,
class                    0.5])
    
```

Naive Bayes (cont)

```

Training a Classifier     Given discrete or
for Discrete and Count  count data
Features

Load libraries           from sklearn.naiv-
                          e_bayes import
                          MultinomialNB

                          from sklearn.fea-
                          ture_extracti-
                          on.text import
                          CountVectorizer

Create bag of words      count = CountV-
                          ectorizer()

                          bag_of_words =
                          count.fit_transfor-
                          m(text_data)

Create feature matrix    features =
                          bag_of_words.to-
                          array()

Create multinomial       classifier = Multin-
naive Bayes object      omialNB(class_p-
with prior probabilities rior=[0.25, 0.5])
of each class

Training a Naive Bayes  Classifier for Binary
Features

Load libraries           from sklearn.naiv-
                          e_bayes import
                          BernoulliNB

Create Bernoulli Naive  classifier =
Bayes object with prior BernoulliNB(cla-
probabilities of each   ss_prior=[0.25,
class                   0.5])
    
```

Naive Bayes (cont)

```

Calibrating             You want to calibrate the
Predicted                predicted probabilities from
Probabilities           naive Bayes classifiers so
                          they are interpretable.

Load libraries          from sklearn.calibration
                          import CalibratedClass-
                          ifierCV

Create                  classifier_sigmoid = Calibr-
calibrated              atedClassifierCV(classifier,
cross-val-              cv=2, method='sigmoid')
idation with
sigmoid
calibration

Calibrate               classifier_sigmoid.fit(fe-
probabilities           atures, target)

View                    classifier_sigmoid.pedic-
calibrated              t_proba(new_observation)
probabilities

If class_prior is not  specified, prior probab-
ilities are learned    using the data. Howev-
er, if we want a uni-  form distribution to be
used as the prior,    we can set fit_prior=False.
    
```

Logistic Regression

```

Training a               from sklearn.linear_model
Binary                  import LogisticRegression
Classifier

                          from sklearn.preproc-
                          essing import StandardS-
                          caler

Create                  logistic_regression =
logistic                LogisticRegression(rando-
regression              m_state=0)
object
    
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 9 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Logistic Regression (cont)

View predicted probabilities

```
model.predict_proba(
new_observation)
```

Training a Multiclass Classifier

Create one-vs-rest logistic regression object

```
logistic_regression =
LogisticRegression(rando
m_state=0, multi_class="
ovr")
```

Reducing Variance Through Regularization

Tune the regularization strength hyperparameter, C

Create decision tree classifier object

```
logistic_regression =
LogisticRegressionCV(
penalty='l2', Cs=10,
random_state=0, n_jobs=-
1)
```

Training a Classifier on Very Large Data

Create logistic regression object

```
logistic_regression =
LogisticRegression(rando
m_state=0, solver="sag")
```

Handling Imbalanced Classes

Create target vector indicating if class 0, otherwise 1

```
target = np.where((target
== 0), 0, 1)
```

Create decision tree classifier object

```
logistic_regression =
LogisticRegression(rando
m_state=0, class_weight=
"balanced")
```

K-Nearest Neighbors

Finding an Observation's Nearest Neighbors

```
from sklearn.neighbors
import NearestNeighbors
```

Create standardizer

```
standardizer = StandardS
caler()
```

Standardize features

```
features_standardized =
standardizer.fit_transfo
rm(features)
```

Two nearest neighbors

```
nearest_neighbors =
NearestNeighbors(n_ne
ighbors=2).fit(features_
standardized)
```

Create an observation

```
new_observation = [ 1, 1,
1, 1]
```

Find distances and indices of the observation's nearest neighbors

```
distances, indices =
nearest_neighbors.kne
ighbors([new_observat
ion])
```

View the nearest neighbors

```
features_standardized[ind
ices]
```

Find two nearest neighbors based on euclidean distance

```
nearestneighbors_eucl
idean = NearestNeigh
bors( n_neighbors=2,
metric='euclidean').fit(fea
tures_standardized)
```

create a matrix indicating each observation's nearest neighbors

K-Nearest Neighbors (cont)

Find each observation's three nearest neighbors based on euclidean distance (including itself)

```
nearestneighbors_e
uclidean = NearestNe
ighbors( n_neighbo
rs=3, metric="eucli
dean").fit(features_st
andardized)
```

List of lists indicating each observation's 3 nearest neighbors

```
nearest_neighbors_
with_self = nearestne
ighbors_euclidean.k
neighbors_graph(
features_standardi
zed).toarray()
```

Remove 1's marking an observation is a nearest neighbor to itself

```
for i, x in enumerate
(nearest_neighbors_
with_self):
```

```
x[i] = 0
```

View first observation's two nearest neighbors

```
nearest_neighbors_
with_self[0]
```

Creating a K-Nearest Neighbor Classifier

Train a KNN classifier with 5 neighbors

```
knn = KNeighborsCl
assifier(n_neighbo
rs=5, n_jobs=-1).fit(-
X_std, y)
```

Identifying the Best Neighborhood Size

Load libraries

```
from sklearn.pipeline
import Pipeline,
FeatureUnion
```

```
from sklearn.model_s
election import
GridSearchCV
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 10 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

K-Nearest Neighbors (cont)

Create a pipeline	<code>pipe = Pipeline([("standardizer", standardizer), ("knn", knn)])</code>
Create space of candidate values	<code>search_space = [{"knn_n_neighbors": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}</code>
Create grid search	<code>classifier = GridSearchCV(pipe, search_space, cv=5, verbose=0).fit(features_standardized, target)</code>
Best neighborhood size (k)	<code>classifier.best_estimator_.get_params()["knn_n_neighbors"]</code>
Creating a Radius-Based Nearest Neighbor Classifier	<code>from sklearn.neighbors import RadiusNeighborsClassifier</code>
Train a radius neighbors classifier	<code>rnn = RadiusNeighborsClassifier(radius=.5, n_jobs=-1).fit(features_standardized, target)</code>

Model Selection

Selecting Best Models Using Exhaustive Search	<code>from sklearn.model_selection import GridSearchCV</code>
Create range of candidate penalty hyperparameter values	<code>penalty = ['l1', 'l2']</code>

Model Selection (cont)

Create range of candidate regularization hyperparameter values	<code>C = np.logspace(0, 4, 10)</code> <code>numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)</code>
Create dictionary hyperparameter candidates	<code>hyperparameters = dict(C=C, penalty=penalty)</code>
Create grid search	<code>gridsearch = GridSearchCV(logistic, hyperparameters, cv=5, verbose=0)</code>
Fit grid search	<code>best_model = gridsearch.fit(features, target)</code>
Predict target vector	<code>best_model.predict(features)</code>
Selecting Best Models Using Randomized Search	
Load libraries	<code>from sklearn.model_selection import RandomizedSearchCV</code>
Create range of candidate regularization penalty hyperparameter values	<code>penalty = ['l1', 'l2']</code>

Model Selection (cont)

Create distribution of candidate regularization hyperparameter values	<code>from scipy.stats import uniform</code> <code>C = uniform(loc=0, scale=4)</code>
Create hyperparameter options	<code>hyperparameters = dict(C=C, penalty=penalty)</code>
Create randomized search	<code>randomizedsearch = RandomizedSearchCV(logistic, hyperparameters, random_state=1, n_iter=100, cv=5, verbose=0, n_jobs=-1)</code>
Fit randomized search	<code>best_model = randomizedsearch.fit(features, target)</code>
Predict target vector	<code>best_model.predict(features)</code>
Selecting Best Models from Multiple	
Load libraries	<code>from sklearn.model_selection import GridSearchCV</code> <code>from sklearn.pipeline import Pipeline</code>
Create a pipeline	<code>pipe = Pipeline([("classifier", RandomForestClassifier())])</code>



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 11 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Model Selection (cont)

Create dictionary with candidate learning algorithms and their hyperparameters

```
search_space = [{"classifier":
  [LogisticRegression()], "classifier__penalty": ['l1', 'l2'], "classifier__C": np.logspace(0, 4, 10)}, {"classifier": [RandomForestClassifier()], "classifier__n_estimators": [10, 100, 1000], "classifier__max_features": [1, 2, 3]}]
```

Create grid search

```
gridsearch = GridSearchCV(
  (pipe, search_space, cv=5,
  verbose=0)
```

Fit grid search

```
best_model = gridsearch.fit(
  features, target)
```

View best model

```
best_model.best_estimator_g-
  et_params()["classifier"]
```

Predict target vector

```
best_model.predict(features)
```

Selecting Best Models When Preprocessing

Load libraries

```
from sklearn.pipeline import
  Pipeline, FeatureUnion
```

Model Selection (cont)

Create a preprocessing object that includes StandardScaler features and PCA

```
preprocess = FeatureUn-
  ion(["std", StandardScal-
  er()), ("pca", PCA())])
```

Create a pipeline

```
pipe = Pipeline(["preproc-
  ess", preprocess), ("class-
  ifier", LogisticRegression-
  ())]])
```

Create space of candidate values

```
search_space = [{"prep-
  rocess__pca__n_compon-
  ents": [1, 2, 3], "classifie-
  r__penalty": ["l1", "l2"], "-
  classifier__C": np.logspa-
  ce(0, 4, 10)}]
```

Create grid search

```
clf = GridSearchCV(pipe,
  search_space, cv=5,
  verbose=0, n_jobs=-1)
```

Fit grid search

```
best_model = clf.fit(featu-
  res, target)
```

Speeding Up Model Selection with Parallelization

```
n_jobs=-1
```

```
gridsearch = GridSearc-
  hCV(logistic, hyperpara-
  meters, cv=5, n_jobs=-1,
  verbose=1)
```

Model Selection (cont)

peeding Up Model Selection Using Algorithm-Specific Methods

If you are using a select number of learning algorithms, use scikit-learn's modelspecific cross-validation hyperparameter tuning.

Create cross-validated logistic regression

```
logit = linear_model.Logis-
  ticRegressionCV(Cs=100)
```

Train model

```
logit.fit(features, target)
```

Evaluating Performance After Model Selection

Load libraries

```
from sklearn.model_sele-
  ction import GridSearchCV,
  cross_val_score
```

Conduct nested cross-validation and output the average score

```
cross_val_score(gridsearch,
  features, target).mean()
```

In scikit-learn, many learning algorithms (e.g., ridge, lasso, and elastic net regression) have an algorithm-specific cross-validation method to take advantage of this.

Handling Dates and Times

Create strings

```
date_strings = np.array(['03-04-
  2005 11:35 PM', '23-05-2010
  12:01 AM', '04-09-2009 09:09
  PM'])
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 12 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Handling Dates and Times (cont)

Convert to datetimes `[pd.to_datetime(date, format='%d-%m-%Y %l:%M %p', errors="coerce") for date in date_strings]`

Handling Time Zones

Create datetime `pd.Timestamp("2017-05-01 06:00:00", tz='Europe/London')`

We can add a time zone to a previously created datetime `date_in_london = date.tz_localize('Europe/London')`

convert to a different time zone `date_in_london.tz_convert('Africa/Abidjan')`

tz_localize and tz_convert to every element `dates.dt.tz_localize('Africa/Abidjan')`

importing all_timezones `from pytz import all_timezones`

Create datetimes range `dataframe['date'] = pd.date_range('1/1/2001', periods=100000, freq='H')`

Select observations between two datetimes `dataframe[(dataframe['date'] > '2002-1-1 01:00:00') & (dataframe['date'] <= '2002-1-1 04:00:00')]`

Breaking Up Date Data into Multiple Features `dataframe['year'] = dataframe['date'].dt.year`

Handling Dates and Times (cont)

`dataframe['month'] = dataframe['date'].dt.month`

`dataframe['day'] = dataframe['date'].dt.day`

`dataframe['hour'] = dataframe['date'].dt.hour`

`dataframe['minute'] = dataframe['date'].dt.minute`

Calculate duration between features `pd.Series(delta.days for delta in (dataframe['Left'] - dataframe['Arrived']))`

Show days of the week `dates.dt.weekday_name`

Show days of the week as numbers (Monday is 0) `dates.dt.weekday`

Creating a Lagged Feature (Lagged values by one row) `dataframe["previous_days_stock_price"] = dataframe["stock_price"].shift(1)`

Calculate rolling mean or moving average `dataframe.rolling(window=2).mean()`

Handling Missing Data in Time Series

Interpolate missing values `dataframe.interpolate()`

Handling Dates and Times (cont)

replace missing values with the last known value (i.e., forward-filling) `dataframe.ffill()`

replace missing values with the latest known value (i.e., backfilling) `dataframe.bfill()`

If we believe the line between the two known points is nonlinear `dataframe.interpolate(method="quadratic")`

Interpolate missing values `dataframe.interpolate(limit=1, limit_direction="forward")`

Handling Numerical Data

Min Max scaler `from sklearn import preprocessing`

Create scaler `minmax_scale = preprocessing.MinMaxScaler(feature_range=(0, 1))`

Scale feature `minmax_scale.fit_transform(feature)`

Standardizing a Feature `from sklearn import preprocessing`

Create scaler `scaler = preprocessing.StandardScaler()`

Transform the feature `standardized = scaler.fit_transform(x)`



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 13 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Handling Numerical Data (cont)

Normalizing Observations (unit norm -> all values have values lower than one)

```
from sklearn.preprocessing import Normalizer
```

Create normalizer

```
normalizer = Normalizer(norm="l2")
```

Transform feature matrix

```
normalizer.transform(features)
```

This type of rescaling is often used when we have many equivalent features (e.g., text classification)

Generating Polynomial and Interaction Features

```
from sklearn.preprocessing import PolynomialFeatures
```

Create PolynomialFeatures object

```
polynomial_interaction = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
```

Create polynomial features

```
polynomial_interaction.fit_transform(features)
```

Transforming Features

```
from sklearn.preprocessing import FunctionTransformer
```

does the same as apply

Handling Numerical Data (cont)

Detecting Outliers

```
from sklearn.covariance import EllipticEnvelope
```

Create detector

```
outlier_detector = EllipticEnvelope(contamination=.1)
```

Fit detector

```
outlier_detector.fit(features)
```

Predict outliers

```
outlier_detector.predict(features)
```

IQR for outlier detection

```
def indices_of_outliers(x):
```

```
    q1, q3 = np.percentile(x, [25, 75])
```

```
    iqr = q3 - q1
```

```
    lower_bound = q1 - (iqr * 1.5)
```

```
    upper_bound = q3 + (iqr * 1.5)
```

```
    return np.where((x > upper_bound) | (x < lower_bound))
```

Handling Outliers

```
houses[houses["Bathrooms"] < 20]
```

Create feature based on boolean condition to detect outliers

```
houses["Outlier"] = np.where(houses["Bathrooms"] < 20, 0, 1)
```

Transform the feature to dampen the effect of the outlier

```
houses["Log_Of_Square_Feet"] = [np.log(x) for x in houses["Square_Feet"]]
```

Handling Numerical Data (cont)

Standardization if we have outliers

```
RobustScaler
```

Discretizing Features (binning)

```
from sklearn.preprocessing import Binarizer
```

Create binarizer

```
binarizer = Binarizer(18)
```

Transform feature

```
binarizer.fit_transform(age) array([[0], [0],
```

```
break up numerical features according to multiple thresholds
np.digitize(age, bins=[20,30,64], right=True (closes the right interval instead of the left))
```

Grouping Observations Using Clustering

```
from sklearn.cluster import KMeans
```

Make k-means clusterer

```
clusterer = KMeans(3, random_state=0)
```

Fit clusterer

```
clusterer.fit(features)
```

Predict values

```
dataframe["group"] = clusterer.predict(features)
```

Keep only observations that are not (denoted by ~) missing

```
features[~np.isnan(features).any(axis=1)]
```

drop missing observations using pandas

```
dataframe.dropna()
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 14 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Handling Numerical Data (cont)

Predict the missing values in the feature matrix

```
features_knn_imputed =
KNN(k=5, verbose=0).c-
omplete(standardized_fea-
tures)
```

Imputer module to fill in missing values

```
from sklearn.preproc-
essing import Imputer
```

Create imputer

```
mean_imputer = Impute-
r(strategy="mean", axis=0)
```

Impute values

```
features_mean_imputed =
mean_imputer.fit_transfo-
rm(features)
```

One option is to use fit to calculate the minimum and maximum values of the feature, then use transform to rescale the feature. The second option is to use fit_transform to do both operations at once. There is no mathematical difference between the two options, but there is sometimes a practical benefit to keeping the operations separate because it allows us to apply the same transformation to different sets of the data.

Deep learning

Preprocessing Data for Neural Networks

Load libraries

```
from sklearn import prepro-
cessing
```

Create scaler

```
scaler = preprocessing.Stan-
dardScaler()
```

Deep learning (cont)

Transform the feature

```
features_standardized =
scaler.fit_transform(features)
# Show feature features_sta-
ndardized array([[ -1.1254-
1308, 1.96429418], [ -1.15-
329466,
```

Designing a Neural Network

Load libraries

```
from keras import models
```

```
from keras import layers
```

Start neural network

```
network = models.Seque-
ntial()
```

Add fully connected layer with a ReLU activation function

```
network.add(layers.Dense-
(units=16, activation="relu",
input_shape=(10,)))
```

Add fully connected layer with a ReLU activation function

```
network.add(layers.Dense-
(units=16, activation="relu"))
```

Add fully connected layer with a sigmoid activation function

```
network.add(layers.Dense-
(units=1, activation="sigm-
oid"))
```

Deep learning (cont)

Compile neural network

```
network.compile(loss="binary-
_crossentropy", # Cross--
entropy optimizer="rmsprop",
# Root Mean Square Propag-
ation metrics=["accuracy"]) #
Accuracy performance metric
```

Training a Binary Classifier

Load libraries

```
from keras.datasets import
imdb
```

```
from keras.preprocessing.text
import Tokenizer
```

```
from keras import models
```

```
from keras import layers
```

Set the number of features we want

```
number_of_features = 1000
```

Start neural network

```
network = models.Sequential()
```

Add fully connected layer with a ReLU activation function

```
network.add(layers.Dense(un-
its=16, activation="relu",
input_shape=( number_of_fe-
atures,)))
```

Add fully connected layer with a ReLU activation function

```
network.add(layers.Dense(un-
its=16, activation="relu"))
```

C

By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 15 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Deep learning (cont)

Add fully connected layer with a sigmoid activation function

```
network.add(layers.Dense(units=1, activation="sigmoid"))
```

Compile neural network

```
network.compile(loss="binary_crossentropy", # Cross-entropy optimizer="rmsprop", # Root Mean Square Propagation metrics=["accuracy"])
```

Train neural network

```
history = network.fit(features_train, # Features target_train, # Target vectorepochs=3, # Number of epochs verbose=1, # Print description after each epoch batch_size=100, # Number of observations per batch validation_data=(features_test, target_test)) # Test data
```

Model Evaluation

Cross-Validating Models

```
from sklearn.model_selection import KFold, cross_val_score
```

```
from sklearn.pipeline import make_pipeline
```

Model Evaluation (cont)

Create a pipeline that standardizes, then runs logistic regression

```
pipeline = make_pipeline(standardizer, logit)
```

Create k-Fold cross-validation

```
kf = KFold(n_splits=10, shuffle=True, random_state=1)
```

Conduct k-fold cross-validation

```
cv_results = cross_val_score(pipeline, # Pipeline features, # Feature matrix target, # Target vector cv=kf, # Cross-validation technique scoring="accuracy", # Loss function n_jobs=-1) # Use all CPU scores
```

Calculate mean

```
cv_results.mean()
```

View score for all 10 folds

```
cv_results
```

Fit standardizer to training set

```
standardizer.fit(features_train)
```

Apply to both training and test sets

```
features_train_std = standardizer.transform(features_train)
```

```
features_test_std = standardizer.transform(features_test)
```

Creating a Baseline Regression Model

```
from sklearn.dummy import DummyRegressor
```

Model Evaluation (cont)

Create a dummy regressor

```
dummy = DummyRegressor(strategy='mean')
```

"Train" dummy regressor

```
dummy.fit(features_train, target_train)
```

Get R-squared score

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

Regression

```
from sklearn.linear_model import LinearRegression
```

Train simple linear regression model

```
ols = LinearRegression()
```

```
ols.fit(features_train, target_train)
```

Get R-squared score

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

Create dummy regressor that predicts 20's for everything

```
clf = DummyRegressor(strategy='constant', constant=20)
```

```
clf.fit(features_train, target_train)
```

Creating a Baseline Classification Model

```
from sklearn.dummy import DummyClassifier
```

Create dummy classifier

```
dummy = DummyClassifier(strategy='uniform', random_state=1)
```

"Train" model

```
dummy.fit(features_train, target_train)
```

Get accuracy score

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



Model Evaluation (cont)

Evaluating Binary Classifier Predictions	<pre>from sklearn.model_selection import cross_val_score from sklearn.datasets import make_classification</pre>
Cross-validate model using accuracy	<pre>cross_val_score(logit, X, y, scoring="accuracy")</pre>
Cross-validate model using precision	<pre>cross_val_score(logit, X, y, scoring="precision")</pre>
Cross-validate model using recall	<pre>cross_val_score(logit, X, y, scoring="recall")</pre>
Cross-validate model using f1	<pre>cross_val_score(logit, X, y, scoring="f1")</pre>
Calculate metrics like accuracy and recall directly	<pre>from sklearn.metrics import accuracy_score</pre>
Calculate accuracy	<pre>accuracy_score(y_test, y_hat)</pre>
Evaluating Binary Classifier Thresholds	<pre>from sklearn.metrics import roc_curve, roc_auc_score</pre>
Get predicted probabilities	<pre>target_probabilities = logit.predict_proba(features_test)[: , 1]</pre>

Model Evaluation (cont)

Create true and false positive rates	<pre>false_positive_rate, true_positive_rate, threshold = roc_curve(target_test, target_probabilities)</pre>
Plot ROC curve	<pre>plt.title("Receiver Operating Characteristic") plt.plot(false_positive_rate, true_positive_rate) plt.plot([0, 1], ls="--") plt.plot([0, 0], [1, 0], c=".7"), plt.plot([1, 1], c=".7") plt.ylabel("True Positive Rate") plt.xlabel("False Positive Rate") plt.show()</pre>
Evaluating Multiclass Classifier Predictions	<pre>cross_val_score(logit, features, target, scoring='f1_macro')</pre>
Visualizing a Classifier's Performance	
libraries	<pre>import matplotlib.pyplot as plt import seaborn as sns from sklearn.metrics import confusion_matrix</pre>

Model Evaluation (cont)

Create confusion matrix	<pre>matrix = confusion_matrix(target_test, target_predicted)</pre>
Create pandas dataframe	<pre>dataframe = pd.DataFrame(matrix, index=class_names, columns=class_names)</pre>
Create heatmap	<pre>sns.heatmap(-dataframe, annot=True, cbar=None, cmap="Blues") plt.title("Confusion Matrix"), plt.tight_layout() plt.ylabel("True Class"), plt.xlabel("Predicted Class") plt.show()</pre>
Evaluating Regression Models	
Cross-validate the linear regression using (negative) MSE cross_val_score	<pre>cross_val_score(ols, features, target, scoring='neg_mean_squared_error')</pre>
Cross-validate the linear regression using R-squared	<pre>cross_val_score(ols, features, target, scoring='r2')</pre>
Evaluating Clustering Models	<pre>from sklearn.metrics import silhouette_score</pre>

Model Evaluation (cont)

	<pre>from sklearn.cluster import KMeans</pre>
Cluster data using k-means to predict classes	<pre>model = KMeans(n_ clusters=2, random_ _state=1).fit(features)</pre>
Get predicted classes	<pre>target_predicted = model.labels_</pre>
Evaluate model	<pre>silhouette_score(fea- tures, target_predicted)</pre>
Creating a Custom Evaluation Metric	<pre>from sklearn.metrics import make_scorer, r2_score</pre>
	<pre>from sklearn.linear_ model import Ridge</pre>
Create custom metric	<pre>def custom_metric(t- arget_test, target_pr- edicted):</pre>
	<pre>r2 = r2_score(target_ _test, target_predi- cted)</pre>
	<pre>return r2</pre>
Make scorer and define that higher scores are better	<pre>score = make_scorer(- custom_metric, greater_is_better=- True)</pre>
Create ridge regression object	<pre>classifier = Ridge()</pre>
Apply custom scorer	<pre>score(model, featur- es_test, target_test)</pre>
Visualizing the Effect of Training Set Size	<pre>from sklearn.model_s- election import learni- ng_curve</pre>

Model Evaluation (cont)

Draw lines	<pre>plt.plot(train_sizes, train_- mean, '--', color="#111111", label="Training score") plt.plot(train_sizes, test_mean, color="#111111", label="Cross-validation score")</pre>
Draw bands	<pre>plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color="#DDDDDD") plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color="#DDDDDD")</pre>
Create plot	<pre>plt.title("Learning Curve") plt.xlabel("Training Set Size"), plt.ylabel("Accuracy Score"), plt.legend(loc="best") plt.tight_layout() plt.show()</pre>
Creating a Text Report of Evaluation Metrics	<pre>from sklearn.metrics import classification_report</pre>

Model Evaluation (cont)

Create a classification report	<pre>print(classification_report(ta- rget_test, target_predicted, target_names=class_na- mes))</pre>
Visualizing the Effect of Hyperparameter Values	
Plot the validation curve	<pre>from sklearn.model_selection import validation_curve</pre>
Create range of values for parameter	<pre>param_range = np.arange(1, 250, 2)</pre>
Hyperparameter to examine	<pre>param_name="n_estimato- rs",</pre>
Range of hyperparameter's values	<pre>param_range = np.arange(1, 250, 2)</pre>



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
 Last updated 9th October, 2022.
 Page 18 of 23.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Model Evaluation (cont)

Calculate accuracy on training and test set using range of parameter values

```
train_scores, test_scores = validation_curve( # Classifier RandomForestClassifier(), # Feature matrix features, # Target vector target, # Hyperparameter to examine param_name="n_estimators", # Range of hyperparameter's values param_range=param_range, # Number of folds cv=3, # Performance metric scoring="accuracy", # Use all computer cores n_jobs=-1)
```

Plot mean accuracy scores for training and test sets

```
plt.plot(param_range, train_mean, label="Training score", color="black")
```

```
plt.plot(param_range, test_mean, label="Cross-validation score", color="dimgrey")
```

Plot accuracy bands for training and test sets

```
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, color="gray")
```

Model Evaluation (cont)

```
plt.fill_between(param_range, test_mean - test_std, test_mean + test_std, color="gainsboro")
```

Create plot

```
plt.title("Validation Curve With Random Forest")
```

```
plt.xlabel("Number Of Trees")
```

```
plt.ylabel("Accuracy Score")
```

```
plt.tight_layout()
```

```
plt.legend(loc="best")
```

```
plt.show()
```

Dimensionality Reduction Using Feature Selection

Thresholding Numerical Feature Variance

```
from sklearn.feature_selection import VarianceThreshold
```

Create thresholder

```
thresholder = VarianceThreshold(threshold=.5)
```

Create high variance feature matrix

```
features_high_variance = thresholder.fit_transform(features)
```

Dimensionality Reduction Using Feature Selection (cont)

View variances

```
thresholder.fit(features).variances_
```

features with low variance are likely less interesting (and useful) than features with high variance.

the VT will not work when feature sets contain different units

If the features have been standardized (to mean zero and unit variance), then for obvious reasons variance thresholding will not work correctly

Handling Text

Strip whitespaces

```
strip_whitespace = [string.strip() for string in text_data]
```

Remove periods

```
remove_periods = [string.replace(".", "") for string in strip_whitespace]
```

Parsing and Cleaning HTML

```
from bs4 import BeautifulSoup
```

Parse html

```
soup = BeautifulSoup(html, "xml")
```

Find the div with the class "full_name", show text

```
soup.find("div", {"class": "full_name"}).text
```

Removing Punctuation

```
import unicodedata
```

```
import sys
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 19 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Handling Text (cont)

Create a dictionary of punctuation characters

```
punctuation = dict.fromkeys(i for i in range(-sys.maxunicode) if unicodedata.category(-chr(i)).startswith('P'))
```

For each string, remove any punctuation characters

```
[string.translate(punctuation) for string in text_data]
```

Tokenizing Text (You have text and want to break it up into individual words)

```
from nltk.tokenize import word_tokenize
```

Tokenize words (string can't have full stops)

```
word_tokenize(string)
```

Tokenize sentences (string has to have full stops)

```
sent_tokenize(string)
```

Removing Stop Words

```
from nltk.corpus import stopwords
```

Load stop words

```
stop_words = stopwords.words('english')
```

Remove stop words

```
[word for word in tokenized_words if word not in stop_words]
```

Stemming Words

```
from nltk.stem.porter import PorterStemmer
```

Create stemmer

```
porter = PorterStemmer()
```

Handling Text (cont)

Apply stemmer

```
[porter.stem(word) for word in tokenized_words]
```

Tagging Parts of Speech

```
from nltk import pos_tag
```

Filter words

```
[word for word, tag in text_tagged if tag in ['NN', 'NNS', 'NNP', 'NNPS']]
```

Tag each word and each tweet

```
for tweet in tweets:
    tweet_tag = nltk.pos_tag(word_tokenize(tweet))
```

```
tagged_tweets.append([tag for word, tag in tweet_tag])
```

Use one-hot encoding to convert the tags into features

```
one_hot_multi = MultiLabelBinarizer()
```

```
one_hot_multi.fit_transform(tagged_tweets)
```

To examine the accuracy of our tagger, we split our text data into two parts

```
from nltk.corpus import brown
```

takes into account the previous two words

```
from nltk.tag import UnigramTagger
```

```
from nltk.tag import BigramTagger
```

```
from nltk.tag import TrigramTagger
```

```
from nltk.tag import UnigramTagger
```

Handling Text (cont)

looks at the word itself

```
from nltk.tag import TrigramTagger
```

Get some text from the Brown Corpus, broken into sentences

```
sentences = brown.tagged_sents(categories='news')
```

Split into 4000 sentences for training and 623 for testing

```
train = sentences[:4000]
test = sentences[4000:]
```

```
test = sentences[4000:]
```

Create backoff tagger

```
unigram = UnigramTagger(train)
```

```
bigram = BigramTagger(train, backoff=unigram)
```

```
trigram = TrigramTagger(train, backoff=bigram)
```

Show accuracy

```
trigram.evaluate(test)
```

Encoding Text as a Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
count = CountVectorizer()
```

Sparse matrix of bag of words

```
bag_of_words = count.fit_transform(text_data)
```

Trun sparse matrix into array

```
bag_of_words.toarray()
```

Show feature (column) names

```
count.get_feature_names()
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 20 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Handling Text (cont)

Create feature matrix with arguments

```
CountVectorizer(ngram_range=(1,2),
stop_words="english", vocabulary=['brasil'])
```

```
bag = count_2gram.fit_transform(text_data)
```

View the 1-grams and 2-grams

```
count_2gram.vocabulary_
```

Weighting Word Importance

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Create the tf-idf (term frequency-document frequency) feature matrix

```
tfidf = TfidfVectorizer()
```

```
feature_matrix = tfidf.fit_transform(text_data)
```

Show feature names

```
tfidf.vocabulary_
```

You will have to download the set of stop words the first time

```
import nltk
nltk.download('stopwords')
```

Note that NLTK's stopwords assumes the tokenized words are all lowercased

Support Vector Machines

Training a Linear Classifier

Load libraries

```
from sklearn.svm import LinearSVC
```

Standardize features

```
scaler = StandardScaler()
```

Support Vector Machines (cont)

```
features_standardized = scaler.fit_transform(features)
```

Create support vector classifier

```
svc = LinearSVC(C=1.0)
```

Train model

```
model = svc.fit(features_standardized, target)
```

Plot data points and color using their class

```
color = ["black" if c == 0 else "lightgrey" for c in target]
```

```
plt.scatter(features_standardized[:,0], features_standardized[:,1], c=color)
```

Create the hyperplane

```
w = svc.coef_[0]
```

```
a = -w[0] / w[1]
```

Return evenly spaced numbers over a specified interval.

```
xx = np.linspace(-2.5, 2.5)
```

```
yy = a * xx - (svc.intercept_[0]) / w[1]
```

Plot the hyperplane

```
plt.plot(xx, yy)
plt.axis("off"),
plt.show()
```

Handling Linearly Inseparable Classes Using Kernels

Create a support vector machine with a radial basis function kernel

```
svc = SVC(kernel="rbf", random_state=0, gamma=1, C=1)
```

Support Vector Machines (cont)

Creating Predicted Probabilities

View predicted probabilities

```
model.predict_proba(new_observation)
```

Identifying Support Vectors

View support vectors

```
model.support_vectors_
```

Handling Imbalanced Classes

Increase the penalty for misclassifying the smaller class using class_weight

Create support vector classifier

```
svc = SVC(kernel="linear", class_weight="balanced", C=1.0, random_state=0)
```

visualization in page 321

In scikit-learn, the predicted probabilities must be generated when the model is being trained. We can do this by setting SVC's probability to True. Then use the same method

Data Wrangling

Creating a series

```
pd.Series(['Molly Mooney', 40, True], index=['Name', 'Age', '-Driver'])
```

Appending to a dataframe

```
dataframe.append(new_person, ignore_index=True)
```

First lines of the data

```
dataframe.head(2)
```

descriptive statistics

```
dataframe.describe()
```

Return row by index

```
dataframe.iloc[0]
```



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 21 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Data Wrangling (cont)

Return row by name	<code>dataframe.loc['Allen, Miss Elisabeth Walton']</code>
Set index	<code>dataframe = dataframe.set_index(dataframe['Name'])</code>
Selecting Rows Based on Conditions	<code>dataframe[dataframe['Sex'] == 'female']</code>
Replacing Values	<code>dataframe['Sex'].replace("anterior", "posterior")</code>
Replacing multiple values	<code>dataframe['Sex'].replace(["female", "male"], ["Woman", "Man"])</code>
Renaming Columns	<code>dataframe.rename(columns={'PClass': 'Passenger Class'})</code>
Minimum, max, sum, count	<code>dataframe['Age'].min()</code>
Finding Unique Values	<code>dataframe['Sex'].unique()</code>
display all unique values with the number of times each value appears	<code>dataframe['Sex'].value_counts()</code>
number of unique values	<code>dataframe['PClass'].nunique()</code>
return booleans indicating whether a value is missing	<code>dataframe[dataframe['Age'].isnull()]</code>

Data Wrangling (cont)

Replace missing values	<code>dataframe['Sex'] = dataframe['Sex'].replace('male', np.nan)</code>
Load data, set missing values	<code>dataframe = pd.read_csv(url, na_values=[np.nan, 'NONE', -999])</code>
Filling missing values	<code>dataframe.fillna(value)</code>
Deleting a Column	<code>dataframe.drop(['Age', 'Sex'], axis=1).head(2)</code>
Deleting a Row	<code>dataframe[dataframe['Sex'] != 'male']</code> or use <code>drop</code>
Dropping Duplicate Rows	<code>dataframe.drop_duplicates()</code>
Dropping Duplicate Rows, taking into account only a subset of rows	<code>dataframe.drop_duplicates(subset=['Sex'], keep='last' (optional argument to keep last observation instead of first))</code>
Grouping Rows by Values	<code>dataframe.groupby('Sex').mean()</code> <code>dataframe.groupby(['Sex', 'Survived'])['Age'].mean()</code>
creating a date range	<code>pd.date_range('06/06/2017', periods=100000, freq='30S')</code>

Data Wrangling (cont)

Group rows by week	<code>dataframe.resample("W").sum()</code>
Group by two weeks	<code>dataframe.resample("2-W").mean()</code>
Group by month	<code>dataframe.resample("M", label='left' (the label returned is the first observation in the group)).count()</code>
Looping Over a Column	<code>for name in dataframe['Name'][0:2]:</code>
Applying a Function Over All Elements in a Column	<code>dataframe['Name'].apply(uppercase)</code>
Applying a Function to Groups	<code>dataframe.groupby('Sex').apply(lambda x: x.count())</code>
Concatenating DataFrames by rows	<code>pd.concat([dataframe_a, dataframe_b], axis=0)</code>
Concatenating DataFrames by columns	<code>pd.concat([dataframe_a, dataframe_b], axis=1)</code>
Merging DataFrames	<code>pd.merge(dataframe_employees, dataframe_sales, on='employee_id', how='left')</code>
	left or right or inner



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 22 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>

Data Wrangling (cont)

```
if the tables have columns with different names
pd.merge(dataframe_employees, dataframe_sales,
         left_on='employee_id',
         right_on='employee_id')
```

replace can accept regular expressions
To have full functionality with NaN we need to import the NumPy library first
groupby needs to be paired with some operation we want to apply to each group, such as calculating an aggregate statistic

Saving and Loading Trained Models

Saving and Loading a scikit-learn Model

```
Load libraries
from sklearn.externals import
joblib
```

```
Save model as pickle file
joblib.dump(model, "model.pkl")
```

```
Load model from file
classifier = joblib.load("model.pkl")
```

```
Get scikit-learn version
scikit_version = joblib.__version__
```

```
Save model as pickle file
joblib.dump(model, "model_{version}.pkl".format(version=scikit_version))
```

Saving and Loading a Keras Model

```
Load libraries
from keras.models import
load_model
```

```
Save neural network
network.save("model.h5")
```

Saving and Loading Trained Models (cont)

```
Load neural network
network = load_model("model.h5")
```

When saving scikit-learn models, be aware that saved models might not be compatible between versions of scikit-learn; therefore, it can be helpful to include the version of scikit-learn used in the model in the filename



By **Remidy08**
cheatography.com/remidy08/

Not published yet.
Last updated 9th October, 2022.
Page 23 of 23.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>