Guía de referencia Scikit-learn

Scikit-learn es una librería de código abierto de Python con una licencia comercialmente usable que implementa una serie de algoritmos de aprendizaje computacional, preprocesamiento, validación cruzada y visualización en una única interfaz construida con *Numpy, Scipy* y *Matplotlib*.

Ejemplo básico

```
from sklearn import tree, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X, y = iris.data[:, :2], iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=33)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
clf = tree.DecisionTreeClassifier(max_depth=3)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

Cargar datos

Los modelos de *scikit-learn* aceptan datos numéricos almacenados en arreglos de *Numpy* o en matrices dispersas de *Scipy*. Otros tipos de datos convertibles a arreglos de *Numpy* también son aceptables, como listas y *DataFrames* de Pandas.

Partición Entrenamiento - Prueba

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Preprocesamiento

Estandarización - media cero y varianza uno

```
scaler = preprocessing.StandardScaler()
X_train_standarized = scaler.fit_transform(X_train)
# NO se hace fit con los datos de prueba
X_test_standarized = scaler.transform(X_test)
```

Normalización - vectores con norma 1

```
# fit no hace nada
scaler = preprocessing.Normalizer()
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

Binarización

```
binarizer = preprocessing.Binarizer(threshold=0.0)
X_binary = binarizer.fit_transform(X)
```

Codificación de variables categóricas

```
enc = preprocessing.OneHotEncoder()
X_train_encoded = enc.fit_transform(X_train)
```

Codificación de etiquetas

```
enc = preprocessing.LabelEncoder()
y = enc.fit_transform(y)
```

Imputación de variables faltantes

```
imp = preprocessing.Imputer(missing_values=0, strategy='mean',
axis=0)
```

Discretización

```
from preprocessing import KBinsDiscretizer
discretizer = KBinsDiscretizer(n_bins=[3, 2, 2], encode='ordinal')
```

Generación de características polinomiales

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(2)
```

Modelos

Aprendizaje supervisado

Regresión lineal

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize=True)
```

Naïve bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

KNN

```
from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(n_neighbors=14)
```

Árboles de decisión

```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='gini')
```

Bosques aleatorios

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=2,
                             random_state=32)
```

Máquinas de Vectores de Soporte (SVM)

```
from sklearn.svm import SVC
svc = SVC(kernel='linear')
```

Aprendizaje no supervisado

Análisis de Componentes Principales (PCA)

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
```

K-Means

```
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters=5, random_state=2)
```

Predicción

Modelos supervisados

```
y_pred = svc.predict(X_test)
y_pred = knn.predict_proba(X_test)
```

Predecir etiquetas
Probabilidad de etiquetas

Modelos no supervisados

```
y_pred = k_means.predict(X_test)
```

Etiqueta en agrupamiento

Pipelines

Ejemplo básico

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([('scaler', StandardScaler()),
                 ('svc', SVC())])
# Un pipeline puede ser usado como cualquier modelo
pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

Ejemplo avanzado

```
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer

num_proc = make_pipeline(
  SimpleImputer(strategy='median'), StandardScaler())
cat_proc = make_pipeline(
  SimpleImputer(strategy='constant',
                fill_value='missing'
  ),
  OneHotEncoder(handle_unknown='ignore'))

# Se aplican transformaciones a cada columna
preprocessor = make_column_transformer(
  (num_proc, ('feat1', 'feat3')),
  (cat_proc, ('feat0', 'feat2')))

clf = make_pipeline(preprocessor, LogisticRegression())
```

Evaluación del desempeño

Métricas para clasificación

Exactitud

```
print(knn.score(X_test, y_test))
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

score de modelos
accurancy_score de metrics

Reporte de Clasificación

```
from sklearn.metrics import classification_report
classification_report(y_test, y_pred)
```

precision, recall, F-1

Matriz de Confusión

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

Métricas para regresión

Error Absoluto Medio

```
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_true, y_pred))
```

Error Cuadrático Medio

```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

Métricas para agrupamientos

Inercia

```
print(k_means_model.inertia_)
```

Coeficiente de Silueta

```
labels = k_means_model.labels_
print(metrics.silhouette_score(X, labels, metric='euclidean'))
```

Homogeneidad

```
from sklearn.metrics import homogeneity_score
print(homogeneity_score(y_true, y_pred))
```

Validación cruzada

```
from sklearn.cross_validation import cross_val_score
print(cross_val_score(knn, X_train, y_train, cv=5))
print(cross_val_score(lr, X, y, cv=4))
```

Afinar modelos

Búsqueda de hiperparámetros - *GridSearch*

```
from sklearn.grid_search import GridSearchCV
params = {"n_neighbors": np.arange(1,9),
          "metric": ["euclidean", "cityblock"]}
grid = GridSearchCV(estimator=knn, param_grid=params)
grid.fit(X_train, y_train)
print(grid.best_score_)
print(grid.best_estimator_.n_neighbors)
```

Búsqueda de Hiperparámetros aleatorizada

```
from sklearn.grid_search import RandomizedSearchCV
params = {"n_neighbors": range(1,10),
          "weights": ["uniform", "distance"]}
rsearch = RandomizedSearchCV(estimator=knn, param_distributions=params,
                             cv=4, n_iter=8, random_state=47)
rsearch.fit(X_train, y_train)
print(rsearch.best_score_)
```