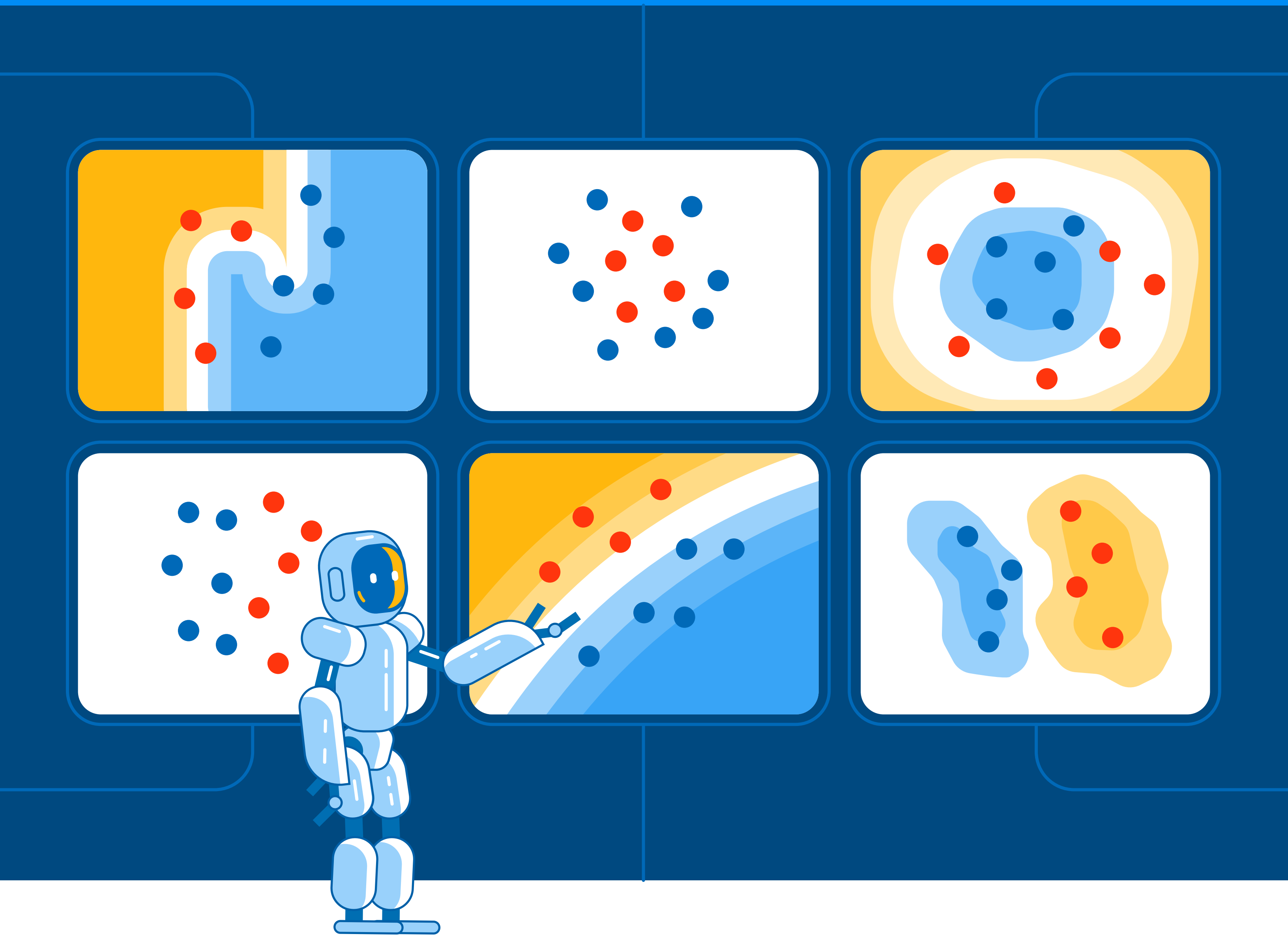
SCIKIT-LEARN

Library for Machine Learning and Data Science with Python



SUPER DATASCIENCE
MAKING THE COMPLEX SIMPLE

WELCOME

Welcome to the **Scikit — Learn** or **Sklearn cheat sheet**. Sklearn is a useful library for Machine Learning and Data Science with Python. Sklearns powers come from the fact that it's extremely simplistic and efficient to build models.

The Sklearn library is built on **NumPy**, **SciPy**, and **Matplotlib**.

It covers a range of operations including :

- Classification
- Clustering
- Regression
- Dimensionality Reduction
- Model Selection
- Preprocessing
- Others...

Dependencies

Scikit-learn requires:

```
Python (>= 2.7 or >= 3.4)
NumPy (>= 1.8.2)
SciPy (>= 0.13.3)
```

Install

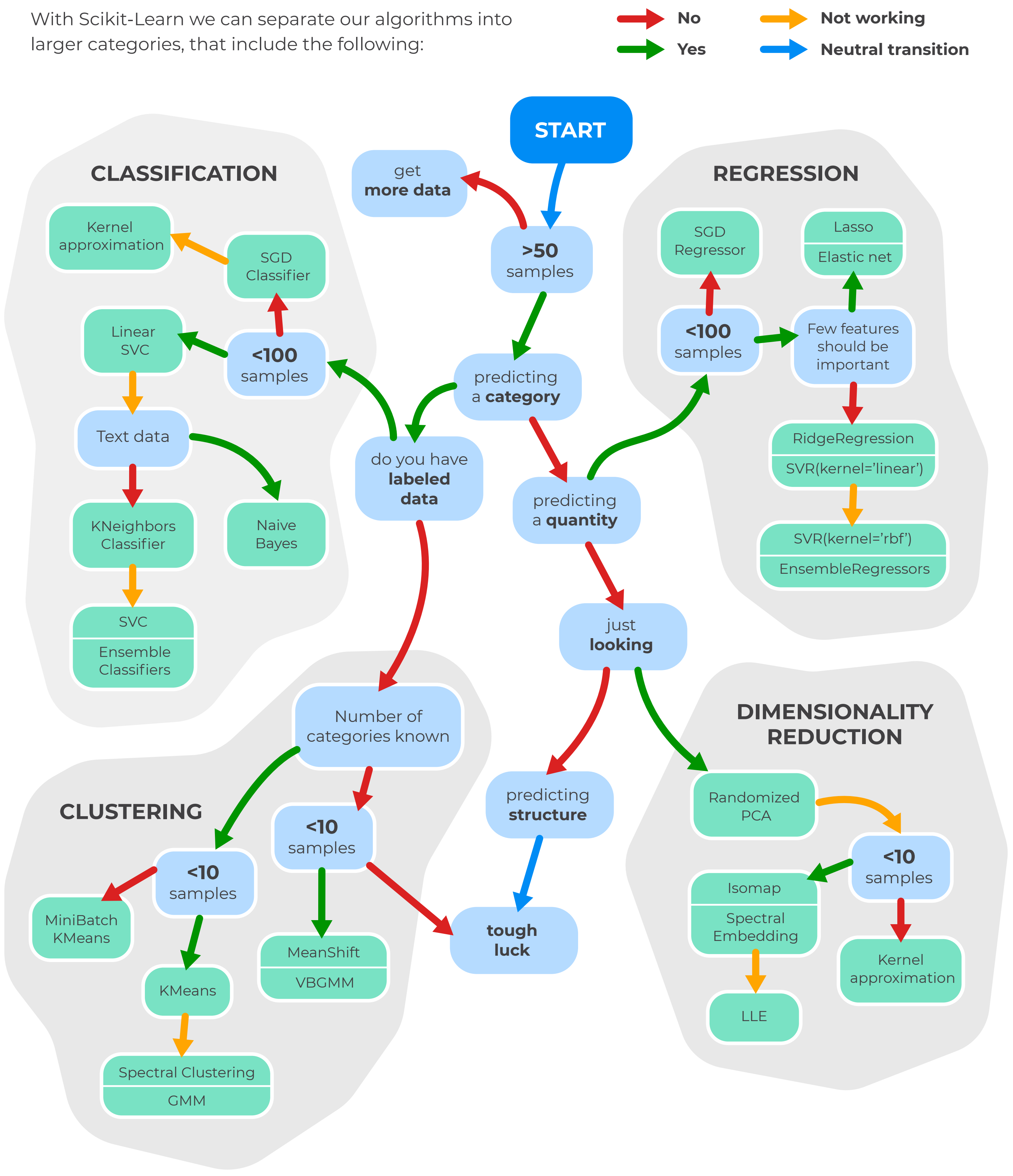
```
pip install -U scikit-learn
```

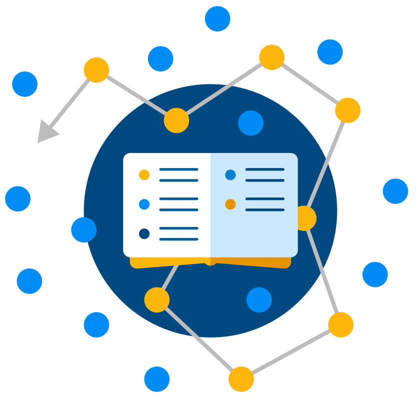
or

```
conda install scikit-learn
```

TYPES OF ALGORITHMS

With Scikit-Learn we can separate our algorithms into larger categories, that include the following:





Supervised Learning

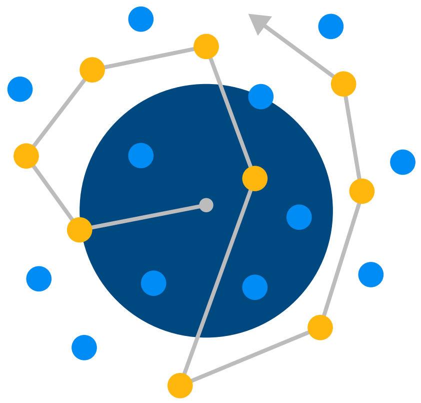
Data comes with additional attributes that we want to predict and can be broken down further into:

Classification

SciKit-Learn provides us with the following write up for Classification stating: Samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data. An example of a classification problem would be the handwritten digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

Regression

Also in the documentation for Sci-kit Learn we can see that: if the desired output consists of one or more continuous variables, then the task is called regression. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.



Unsupervised Learning

Training data consists of a set of input vectors x without any corresponding target values or unlabeled data.

Clustering

Clustering — can be used to determine the distribution of data within the input space, also referred to as density estimation.

TRAINING SET AND TESTING SET

Machine learning is about learning some properties of a data set and applying them to new data. This is why a common practice in machine learning to evaluate an algorithm is to split the data at hand into two sets, one that we call the training set on which we learn data properties and one that we call the testing set on which we test these properties.

Train Test Split

```
import numpy as np
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0)
```

Random state

Scikit-learn does not use its own global random state; random_state can use **numpy global random state** that can be set using: **numpy.random.seed**

```
import numpy as np
np.random.seed(20)
```

Standardization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
standardized_X = scaler.transform(X_train)
standardized_X_test = scaler.transform(X_test)
```

Normalization

```
from sklearn.preprocessing import Normalizer
normalizer = preprocessing.Normalizer().fit(X)
# fit does nothing normalizer
```

Binarization

```
from sklearn.preprocessing import Binarizer
binarizer = preprocessing.Binarizer(threshold=1.1)
binarizer.transform(X)
```

Scaling Data

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

WORKING WITH DATA

Using **Numpy** or **Pandas**

Training or Testing Data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

# random_state simply sets a seed to the random generator, so that your train-test splits are always deterministic

Model Fitting

Supervised learning

```
lr.fit(X, y)
knn.fit(X_train, y_train)
svc.fit(X_train, y_train)
k_means.fit(X_train)
pca_model = pca.fit_transform(X_train)
```

Prediction

Supervised Estimators

```
y_pred = svc.predict(np.random.random((1,9)))
y_pred = lr.predict(X_test)
y_pred = knn.predict_proba(X_test)
y_pred = k_means.predict(X_test)
```

Unsupervised Estimators — you can remove this

Imputing Missing Values

Imputation transformer for completing missing values and can be replaced with the mean, the median or the most frequent value.

```
from sklearn.preprocessing import Imputer
build_imputer = Imputer(missing_values='NaN', strategy='mean', axis=0)
build_imputer = imp.fit(X_train)
```

OneHotEncoding

```
sklearn.preprocessing.OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
```

Log Loss

```
sklearn.metrics.log_loss
```

Make_classification

Generate a random n-class classification problem

```
from sklearn.datasets import make_classification
```

Datasets https://github.com/scikit-learn/scikit-learn/tree/master/sklearn/datasets/data

SETTING UP THE MODELS WITH EXAMPLES

The following walkthroughs are simple steps of importing the models and a few examples for further detail. It's an easy way to prototype and test to help get started but as always it's highly recommended to pull up the parameters for each specific model from the Sklearn documentation depending on the necessities of the algorithm.

1. Classification

Using classification methods to identify the category an object belongs to

SVM

The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input

from **sklearn.svm** import SVC **#SVC** and **NuSVC** implement the “one-against-one” approach

```
model_name = SVC(kernel='linear')
model_name.fit(X_train, y_train)
```

NearestNeighbors

The classes in sklearn.neighbors can handle either Numpy arrays or scipy.sparse matrices as input

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
```

Decision Tree

```
from sklearn import tree
X = [[1, 0], [2, 0]]
Y = [1, 1]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

2. Clustering

K-means

```
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters=3, random_state=0)
```

# setting the number of clusters to form as well as the number of centroids to generate, **default is 8**

3. Regression

Linear Regression - LR

```
from sklearn.linear_model import LinearRegression
model_name = LinearRegression()
```

Fit and Predict - LR

```
model_name.fit(x_training_data, y_training_data)
predictions = model_name.predict(x_data)
```

SVR - Support Vector Regression

```
from sklearn.svm import SVR
import numpy as np

n_samples, n_features = 10, 5
np.random.seed(0)
y = np.random.randn(n_samples)
X = np.random.randn(n_samples, n_features)
clf = SVR(C=1.0, epsilon=0.2)
clf.fit(X, y)
```

The support vector machines in the sklearn library state support for: both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an **SVM** to make predictions for sparse data, it must have been fit on such data.
For optimal performance, use **C-ordered numpy.ndarray (dense) orscipy.sparse.csr_matrix (sparse)** with **dtype=float64.**

```
from sklearn import svm
```

Ridge Regression

Linear least squares with l2 regularization.

```
from sklearn.linear_model import Ridge
```

4. Dimensionality Reduction

PCA - Principal Component Analysis

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

```
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[1, -1], [-3, -1], [-1, -1], [2, 2], [-1, 1], [1, 0]])
pca = PCA(ncomponents=2)
pca.fit(X)
print(X)
print(pca.explained_variance_ratio)
print(pca.singularvalues)
```

5. Model Selection

Grid Search

from sklearn.model_selection import GridSearchCV

Cross-Validation

The easiest method of using cross-validation in Sklearn is to call the cross_val_score helper function on the estimator and the dataset. The example below shows how to estimate the accuracy of a linear kernel support vector machine by splitting the data, fitting a model and computing the **score X (5)** consecutive times (with different splits each time) http://scikit-learn.org/stable/modules/cross_validation.html

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

6. Pre Processing

Pre-Processing

Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

```
from sklearn import preprocessing
import numpy as np
X_trsin = np.array([[2,7,1],[4,1,0],[1,1,1]])
X_scaled = preprocessing.scale(X_train)
X_scaled
```

Accuracy Score

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

Confusion Matrix

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

VALIDATING THE MODEL

The **sklearn.metrics** module includes score functions, performance metrics and pairwise metrics and distance computations.

Import and print accuracy, recall, precision, and F1 score:

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
print(accuracy_score(true_labels, guesses)) print(recall_score(true_labels, guesses))
print(precision_score(true_labels, guesses)) print(f1_score(true_labels, guesses))
```

HELPFUL INFORMATION

Citing scikit-learn

If you use scikit-learn in a **scientific publication**, we would appreciate citations to the following paper:

```
Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Bibtex entry:

    @article{scikit-learn,

    title={Scikit-learn: Machine Learning in {P}ython},

    author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion,
            B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg,
            V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and
            Perrot, M. and Duchesnay, E.},

    journal={Journal of Machine Learning Research},

    volume={12},

    pages={2825--2830},

    year={2011}

    }
```

If you want to cite scikit-learn for its **API or design**, you may also want to consider the following paper:

```
API design for machine learning software: experiences from the scikit-learn project,
Buitinck et al., 2013.

Bibtex entry:

    @inproceedings{sklearn_api,

    title={{API} design for machine learning software: experiences from the scikit-learn
    project},

    author={Lars Buitinck and Gilles Louppe and Mathieu Blondel and Fabian Pedregosa and
            Andreas Mueller and Olivier Grisel and Vlad Niculae and Peter Prettenhofer and
            Alexandre Gramfort and Jaques Grobler and Robert Layton and Jake VanderPlas and
            Arnaud Joly and Brian Holt and Ga{\"{e}}l Varoquaux},

    booktitle={ECML PKDD Workshop: Languages for Data Mining and Machine Learning},

    pages = {108--122},

    year={2013}

    }
```

Source: http://scikit-learn.org/stable/index.html

Map: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

API Reference: http://scikit-learn.org/stable/modules/classes.html#api-reference