>>> Introduction to Data Science with Python
>>> DS101

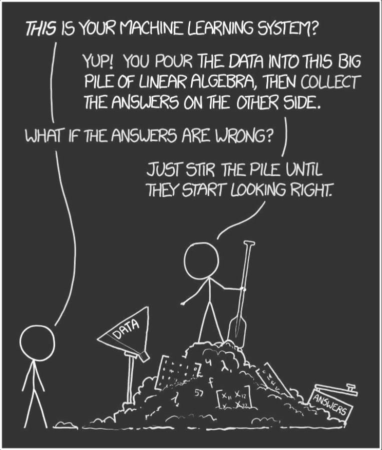
Name: Celia Cintas[†] Nahuel Defosse[‡]

Date: April 6, 2019

[†]cintas.celia@gmail.com
[‡]nahuel.defosse@gmail.com

>>> What is a learning problem?

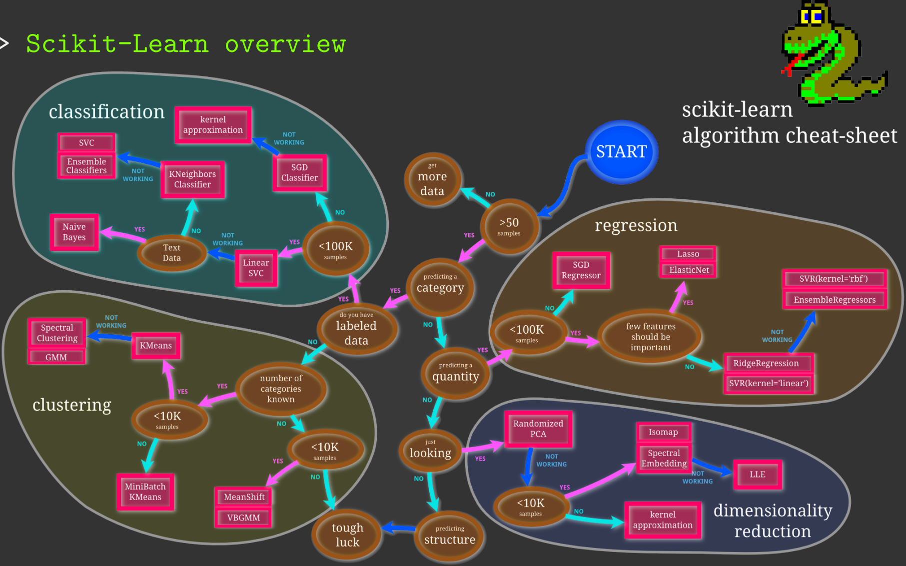
A learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (multivariate data), it is said to have several attributes or features.



>>> What types of problems do we have?

* supervised learning, in which the data comes with additional attributes that we want to predict. This problem can be either:
  * classification: samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data.
  * regression: if the desired output consists of one or more continuous variables, then the task is called regression.
* unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization.

>>> Scikit-Learn overview



>>> Preprocessing our Data

Do I have missing values? How are they expressed in the data? Should I withhold samples with missing values? Or should I replace them? If so, which values should they be replaced with?

```
from sklearn.impute import MissingIndicator
from sklearn.impute import SimpleImputer

X.replace({999.0 : np.NaN}, inplace=True)
indicator = MissingIndicator(missing_values=np.NaN)
indicator = indicator.fit_transform(X)
indicator = pd.DataFrame(indicator, columns=['m1', 'm3'])

imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit_transform(X)
```

>>> Preprocessing our Data (Cont.)

Munging categorical data is another essential process during data preprocessing. It is necessary to convert categorical features to a numerical representation. Is the feature ordinal or nominal?

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
X = pd.DataFrame( np.array(['M', 'O-', 'medium', 'M', 'O-', 'high',
                            'F', 'O+', 'high', 'F', 'AB', 'low',
                            'F', 'B+', np.NaN]).reshape((5,3)))
X.columns = ['sex', 'blood_type', 'edu_level']
enc = OrdinalEncoder(categories=['low', 'medium', 'high'])
X.edu_level = enc.fit_transform(X.edu_level.values.reshape(-1, 1))
onehot = OneHotEncoder(dtype=np.int, sparse=True)
nominals = pd.DataFrame(onehot.fit_transform(X[['sex', 'blood_type']]
                        \.toarray(),
                        columns=['F', 'M', 'AB', 'B+','O+', 'O-'])
nominals['edu_level'] = X.edu_level
```

>>> Preprocessing our Data (Cont.)

Numerical features can be 'decoded' into categorical features. The two most common ways to do this are discretization and binarization.

* Discretization, divides a continuous feature into a pre-specified number of categories (bins).
* Feature binarization is the process of tresholding numerical features to get boolean values.

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

disc = KBinsDiscretizer(n_bins=3, encode='onehot',
                        strategy='uniform')
disc.fit_transform(X)
binarizer = Binarizer(threshold=0, copy=True)
binarizer.fit_transform(X.f3.values.reshape(-1, 1))
```

>>> Preprocessing our Data (Cont.)

The next logical step in our preprocessing pipeline is to scale our features.

* Standard Scaler. $x_{scaled} = (x - u)/s$ Centers the data by using the following formula, where u is the mean and s is the sd.
* MinMax Scaler. Transforms features by scaling each feature to a given range. $x_{scaled} = (x - min(x))/(max(x) - min(x))$.
* Robust Scaler.

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
```

Check out Custom transformations, other Scalers at https://scikit-learn.org/stable/modules/preprocessing.html

>>> Preprocessing our Data (Cont.)

Normalization is the process of scaling individual samples to have unit norm. In basic terms you need to normalize data when the algorithm predicts based on the weighted relationships formed between data points.

One of the key differences between scaling (e.g. standardizing) and normalizing, is that normalizing is a row-wise operation, while scaling is a column-wise operation.

>>> How my data looks like?

one sample

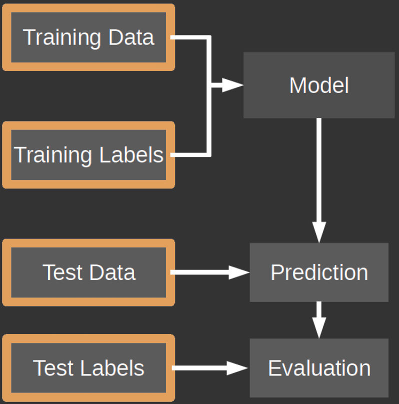
$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix} \qquad y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

one feature

outputs / labels

https://speakerdeck.com/amueller/advanced-machine-learning-with-scikit-learn

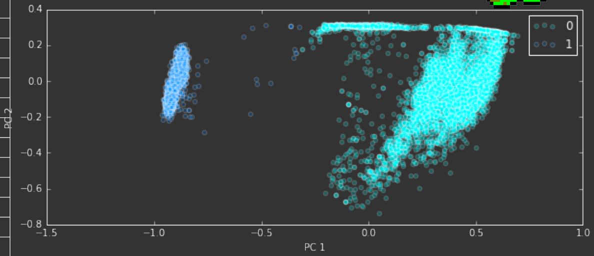
>>> Split your data

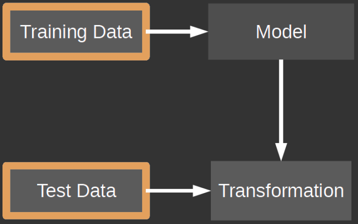



```python
X_train, X_test, \
y_train, y_test = \
train_test_split(Xp, y,
                test_size=0.3,
                random_state=42)
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
clf.score(X_test, y_test)
```

>>> Dimension Reduction

	banda 1	banda 2	banda 3	banda 4	banda 5	banda 7	salida
0	0.839	0.918	0.925	0.643	0.808	0.902	0
1	0.937	0.918	0.902	0.663	0.843	0.902	0
2	0.906	0.835	0.902	0.631	1.000	1.000	0
3	0.969	0.835	0.925	0.631	1.000	1.000	0
4	0.969	0.835	0.925	0.631	1.000	1.000	0
5	0.937	0.835	0.925	0.631	1.000	1.000	0
6	0.839	0.835	0.925	0.631	1.000	1.000	0
7	0.937	0.792	0.953	0.608	1.000	1.000	0
8	0.937	0.961	0.925	0.608	1.000	1.000	0
9	1.000	0.875	0.953	0.608	1.000	1.000	0
10	0.969	0.835	0.925	0.608	1.000	1.000	0
11	1.000	0.875	0.976	0.608	1.000	0.969	0
12	1.000	0.875	0.925	0.620	1.000	1.000	0
13	0.937	0.875	0.925	0.620	1.000	0.984	0
14	1.000	0.918	0.925	0.631	1.000	1.000	0
15	0.937	0.918	0.953	0.631	1.000	1.000	0
16	0.969	0.875	0.953	0.631	1.000	1.000	0
17	1.000	0.918	0.925	0.631	1.000	1.000	0
18	0.937	0.875	0.953	0.643	0.984	1.000	0
19	0.937	0.875	0.902	0.631	0.945	1.000	0
20	0.906	0.835	0.902	0.663	0.933	0.937	0

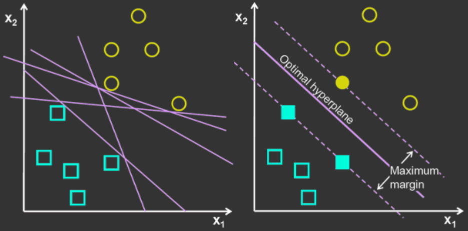





```
pca = PCA(n_components=3)
pca.fit(X_train)
X_new = pca.transform(X_test)
```

>>> Support Vector Machine

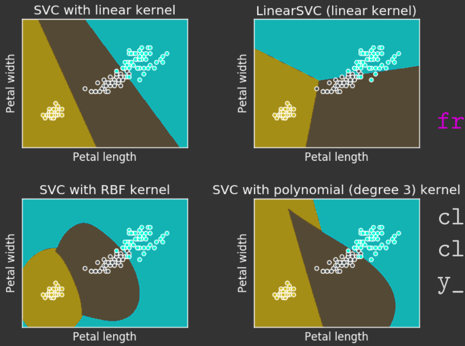
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N - the number of features) that distinctly classifies the data points.



To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.

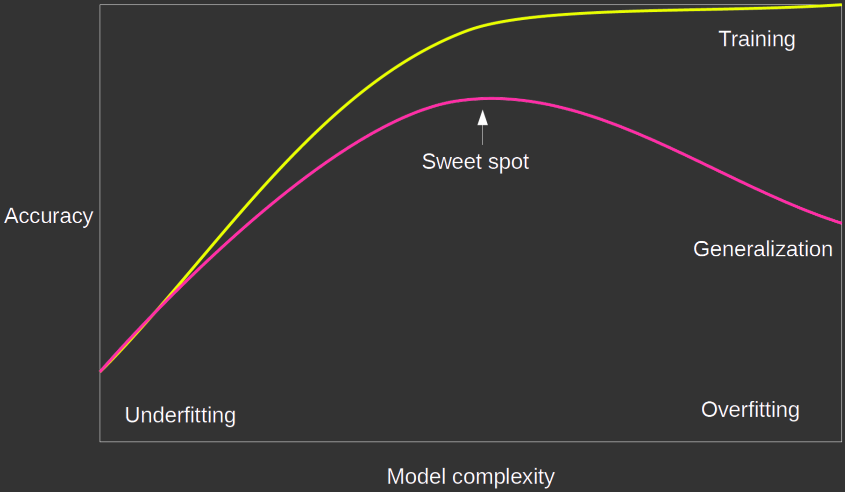
>>> Support Vector Machine (Cont.)

Some problems can' t be solved using linear hyperplane.

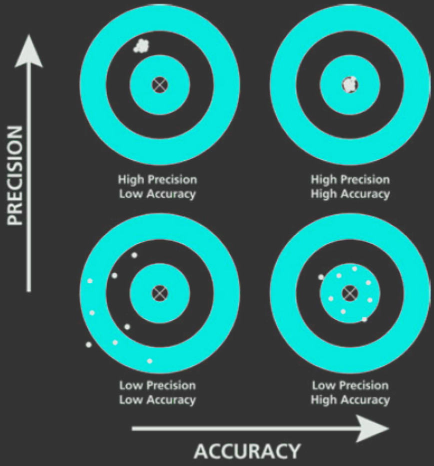


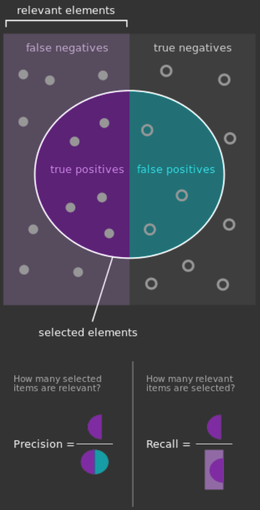

```
from sklearn import svm

clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

>>> Overfitting and Underfitting

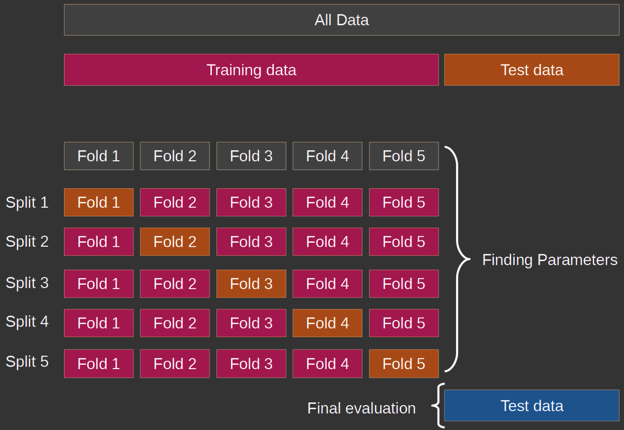


>>> Classification metrics





>>> Cross-validation




```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X, y, cv=5)
print(scores)

>> [ 0.92  1.     1.     1.     1.   ]
```

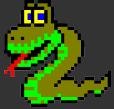
>>> Exercises

preprocessing Load dataset and Scale values between [0, 1].

model Create baseline model and a SVM for classification with dataset provided.

persistence Save model to pickle.

>>> Things to explore & Gracias!



* Scikit-Learn Documentation
  https://scikit-learn.org/stable/tutorial/basic/tutorial.html
* Preprocessing with Sklearn https://towardsdatascience.com/
  preprocessing-with-sklearn-a-complete-and-comprehensive-guide-67
* Hyper-parameter search with Sklearn
  https://scikit-learn.org/stable/modules/grid_search.html
* SVM https://towardsdatascience.com/
  support-vector-machine-introduction-to-machine-learning-algorith
* SciPy Tutorials and talks
  https://www.youtube.com/user/EnthoughtMedia/playlists
* More examples with sklearn https://github.com/celiacintas/
  scipyla2016_tutorials/tree/master/sklearn-intro