Introduction to Machine Learning Cheatsheet

Useful resources:
- scikit-learn
- pandas
- seaborn

Loading data

Your data needs to be tabular and stored as NumPy arrays or compatible format such as Pandas DataFrame.

```
import pandas as pd
df = pd.read_csv("my_data.csv")
```

Preprocessing

Log-transform

With some data (eg, proteomics, RNAseq) log-transform can sometimes help stabilize the data's mean-variance relationship.

```
dflog = np.log2( df )
```

Imputation

Missing data (NA) can be blocking for many ML methods

```
# removing all samples containing NAs:
dfnoNA = df.loc[ df.isna().sum(axis=1)>0 , :  ]


# mean imputation:
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values=0, strategy='mean', axis=0)
X_imputed = imp.fit_transform(X)
```

One-hot encoding

categorical data can be blocking for many ML methods

```
df_encoded = pd.get_dummies( df , drop_first = True )
```

Standardization/scaling

Methods relying on distances (KNN, Kmeans,...) or building weighted combination of features (linear models, PCA, ...) must have properly scaled features.

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

Unsupervised Learning

PCA - NEEDS SCALING

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X_pca = pca.fit_transform(X)
```

Kmeans - NEEDS SCALING

```
from sklearn.cluster import Kmeans
km = Kmeans(n_clusters = 5)
km.fit(X)
km.labels_
```

Hierarchial Clustering - NEEDS SCALING

```
from sklearn.cluster import AgglomerativeClustering
hs = AgglomerativeClustering(n_clusters = 5,
linkage='ward')
hs.fit(X)
hs.labels_
```

Scoring clusters

```
from sklearn.metrics import adjusted_rand_score, silhouette_score
adjusted_rand_score(km.labels_ , hs.labels_)
silhouette_score(X, km.labels_)
```

ML routine

train/test split

train set: find and train the best model

test set: final evaluation of generalizing performance

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

K-fold cross-validation

Useful to get an estimate of a model generalizability.

Typically used to compare models or tune hyper-parameters

```
from sklearn.cross_validation import cross_val_score
print(cross_val_score(model1, X_train, y_train, cv=5))
print(cross_val_score(model2, X_train, y_train, cv=5))
```

Pipeline

```
from sklearn.pipeline import Pipeline
ppl = Pipeline( [('imputer', Imputer(strategy='mean')),
                 ('scaler', StandardScaler()),
                 ('model', KNeighborsClassifier())] )
```

Hyper-parameter optimization with a grid search algorithm

```
from sklearn.grid_search import GridSearchCV
params = {"model__n_neighbors": np.arange(1,3),
                 "imputer__strategy": ['mean','median'] }
grid = GridSearchCV(estimator=ppl,
                                    param_grid=params,
                                    cv = 8,
                                    scoring = 'roc_auc')
grid.fit(X_train, y_train)
print(grid.best_score_)
print(grid.best_params_)
```

Classification

K-Nearest Neighbors - NEEDS SCALING.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5, weights='uniform')
```

Logistic Regression - NEEDS SCALING. INTERPRETABLE.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression( penalty = 'l2', C=1.0 )
```

Decision Tree - no scaling or imputation. INTERPRETABLE.

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=None,
                               min_samples_split=2,
                               min_samples_leaf=1)
```

Random Forest - no scaling or imputation. INTERPRETABLE.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier( n_estimator = 100,
                                max_features='sqrt',
                                max_depth=None,
                                min_samples_split=2,
                                min_samples_leaf=1)
```

Classification scores

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
y_pred = model.predict(X)
y_pred_score = model.decision_function(X)
accuracy_score(y, y_pred)       # <- sensitive to imbalance
f1_score(y, y_pred)             # <- OK when there is imbalance
roc_auc_score(y, y_pred_score)  # <- OK when there is imbalance
```

Regression

Linear Regression - NEEDS SCALING

```
from sklearn.linear_model import SGDRegressor
model = SGDRegressor( penalty='elasticnet',
                      alpha=0.0001,
                      l1_ratio=0.15)
```

Random Forest - no scaling or imputation.

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor( n_estimator = 100,
                               max_features=1.0,
                               max_depth=None,
                               min_samples_split=2,
                               min_samples_leaf=1)
```

Regression scores

```
from sklearn.metrics import r2_score, mean_squared_error
y_pred = model.predict(X)
r2_score(y, y_pred)
mean_squared_error = r2_score(y, y_pred)
```

Made at the Swiss Institute of Bioinformatics
Find out about us at www.sib.swiss