Introduction to Machine Learning with Scikit-Learn

District Data Labs

These slides are based on:
https://www.slideshare.net/BenjaminBengfort/introduction-to-machine-learning-with-scikitlearn
by Benjamin Bengfort

and
by Gabriel Moreira
"Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data."

- Jason Brownlee
“Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering.”

- Andrew Ng
The Dream...

Raw data → Dataset → Model → Task
The Reality

Raw data → Features → ML Ready dataset → Model → Task
... The Reality

ML Ready dataset

Model

Deep Learning

Task

Raw data

But what architecture?

Scikit-learn can help a lot around here
What is Scikit-Learn?

Extensions to SciPy (Scientific Python) are called SciKits. SciKit-Learn provides machine learning algorithms and much more.

- Algorithms for supervised & unsupervised learning
- Built on SciPy and Numpy
- Standard Python API interface
- Sits on top of c libraries, LAPACK, LibSVM, and Cython
- Open Source: BSD License (part of Linux)

Probably the best general ML framework out there.
More on scikit-learn

- Data loading and pre-processing
- Feature engineering
- Hyperparameter tuning and Model evaluation
- Pipelines
Data Loading and pre-processing

- **Data Preparation / pre-processing** (this is where the magic comes😊) [scikit-learn (numpy, Pandas)]
  - Data cleaning, inputs for missing values, features normalization, outliers detection
  - Features augmentation / selection / construction / reduction
    - Transformers (fit -> transform)

- **Exploration (iterates with pre-processing)** [scikit-learn / matplotlib / Seaborn…]
  - Get some insights on the data, understand its structure, create initial hypothesis
  - Clustering algorithms, histograms, plots, correlation…
  - [https://seaborn.pydata.org/generated/seaborn.heatmap.html](https://seaborn.pydata.org/generated/seaborn.heatmap.html)
  - [https://seaborn.pydata.org/generated/seaborn.pairplot.html](https://seaborn.pydata.org/generated/seaborn.pairplot.html)
  - [https://seaborn.pydata.org/generated/seaborn.FacetGrid.html](https://seaborn.pydata.org/generated/seaborn.FacetGrid.html)
  - Visualization (matplotlib, seaborn,…)
Data pre-processing and transformation

- Inputing for NaNs
- Normalization / standardization
- Transform nominals
- Transform numeric
- Data Aggregation
- Feature extraction / combination

Mostly Transformer design patterns
- Fit on train data
- Transform both train and test data
Load dataset

- `sklearn.datasets`
  - Some "toy examples" ready to use
    - `datasets.load_iris()`
    - `datasets.load_wine()`
  - Generate artificial data

```python
>>> from sklearn.datasets import load_digits
>>> digits = load_digits()
>>> digits.data.shape
(1797, 64)
>>> digits.target_names
[0 1 2 3 4 5 6 7 8 9]
>>> import matplotlib.pyplot as plt
>>> plt.gray()
>>> plt.matshow(digits.images[0])
>>> plt.show()
```
Imputation for missing values

- Datasets contain missing values, often encoded as blanks, NaNs or other placeholders
- Ignoring rows and/or columns with missing values is possible, but at the price of losing data which might be valuable
- Better strategy is to infer them from the known part of data
- Strategies
  - Mean: Basic approach
  - Median: More robust to outliers
  - Mode: Most frequent value
  - Using a model: Can expose algorithmic bias
Imputation for missing values

```python
>>> import numpy as np
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
Imputer(axis=0, copy=True, missing_values='NaN', strategy='mean', verbose=0)
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[ 4.   2.]
 [ 6. 3.666...]
 [ 7.  6.]]
```

Missing values imputation with scikit-learn
**Binarization**

- Transform discrete or continuous numeric features in binary features

Example: Number of user views of the same document

```python
>>> from sklearn import preprocessing
>>> X = [[1., -1., 2.],
      ... [2., 0., 0.],
      ... [0., 1., -1.]]

>>> binarizer = preprocessing.Binarizer(threshold=1.0)
>>> binarizer.transform(X)
array([[1., 0., 1.],
      [1., 0., 0.],
      [0., 1., 0.]])
```

Binarization with scikit-learn
Binning

- Split numerical values into bins and encode with a bin ID
- Can be set arbitrarily or based on distribution
- **Fixed-width binning**

Does fixed-width binning make sense for this long-tailed distribution?

```python
>>> x = np.array([0.2, 6.4, 3.0, 1.6])
>>> bins = np.array([0.0, 1.0, 2.5, 4.0, 10.0])
>>> inds = np.digitize(x, bins)
>>> inds
array([1, 4, 3, 2])
```

Most users (458,234,809 ~ 5*10^8) had only 1 pageview during the period.

`numpy.digitize`
**Binning**

- **Adaptative or Quantile binning**
  Divides data into equal portions (eg. by median, quartiles, deciles)

```python
>>> deciles = dataframe['review_count'].quantile([.1, .2, .3, .4, .5, .6, .7, .8, .9])
>>> deciles
0.1 3.0
0.2 4.0
0.3 5.0
0.4 6.0
0.5 8.0
0.6 12.0
0.7 17.0
0.8 28.0
0.9 58.0
```
Scaling

- Models that are smooth functions of input features (or utilize similarity) are sensitive to the scale of the input (e.g., Linear Regression, KNN)
- Scale numerical variables into a certain range, dividing values by a normalization constant (no changes in single-feature distribution)

- Popular techniques
  - MinMax Scaling
  - Standard (Z) Scaling
  - L2 normalization
Min-max scaling

- Squeezes (or stretches) all values within the range of [0, 1] to add robustness to very small standard deviations and preserving zeros for sparse data.

```python
>>> from sklearn import preprocessing
>>> X_train = np.array([[1., -1., 2.],
...                      [2., 0., 0.],
...                      [2., 0., 0.],
...                      [0., 1., -1.]]
...                      ...

>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> print(X_train_minmax)
array([[0.5   , 0.    , 1.    ],
       [1.    , 0.5   , 0.33333333],
       [0.    , 1.    , 0.    ]])
```

Min-max scaling with scikit-learn
Standard (Z) Scaling

After Standardization, a feature has mean of 0 and variance of 1 (assumption of many learning algorithms)

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X = np.array([[[1., -1., 2.], [2., 0., 0.], ...
>>>                [0., 1., -1.]]])
>>> X_scaled = preprocessing.scale(X)
>>> X_scaled
array([[[0.  ..., -1.22..., 1.33...],
        [1.22..., 0.  ..., -0.26...],
        [-1.22..., 1.22..., -1.06...]]])
>>> X_scaled.mean(axis=0)
array([0., 0., 0.])
>>> X_scaled.std(axis=0)
array([1., 1., 1.])
```

Standardization with scikit-learn
Normalization

- Scales individual samples (rows) to have unit vector, dividing values by vector’s $L^2$ norm, a.k.a. the Euclidean norm
- Useful for quadratic form (like dot-product) or any other kernel to quantify similarity of pairs of samples. This assumption is the base of the Vector Space Model often used in text classification and clustering contexts.

\[
\tilde{x} = \frac{x}{\|x\|_2}.
\]  
Normalized vector

\[
\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_m^2}
\]  
Euclidean ($L^2$) norm
from sklearn import preprocessing

X = [[1., -1., 2.],
     [2., 0., 0.],
     [0., 1., -1.]]

X_normalized = preprocessing.normalize(X, norm='l2')

X_normalized

array([[ 0.40..., -0.40...,  0.81...],
        [ 1. ...,  0. ...,  0. ...],
        [ 0. ...,  0.70..., -0.70...]])
Log transformation
Compresses the range of large numbers and expand the range of small numbers.
Eg. The larger x is, the slower log(x) increments.

<table>
<thead>
<tr>
<th>user_id</th>
<th>views_count</th>
<th>log(1+views_count)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1000</td>
<td>6.91</td>
</tr>
<tr>
<td>b</td>
<td>500</td>
<td>6.22</td>
</tr>
<tr>
<td>c</td>
<td>300</td>
<td>5.71</td>
</tr>
<tr>
<td>d</td>
<td>200</td>
<td>5.30</td>
</tr>
<tr>
<td>e</td>
<td>150</td>
<td>5.02</td>
</tr>
<tr>
<td>f</td>
<td>100</td>
<td>4.62</td>
</tr>
<tr>
<td>g</td>
<td>70</td>
<td>4.26</td>
</tr>
<tr>
<td>h</td>
<td>50</td>
<td>3.93</td>
</tr>
<tr>
<td>i</td>
<td>30</td>
<td>3.43</td>
</tr>
<tr>
<td>j</td>
<td>20</td>
<td>3.04</td>
</tr>
<tr>
<td>k</td>
<td>10</td>
<td>2.40</td>
</tr>
<tr>
<td>l</td>
<td>5</td>
<td>1.79</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
<td>0.69</td>
</tr>
</tbody>
</table>
Log transformation

Smoothing long-tailed data with log

Histogram of # views by user

Histogram of # views by user
smoothed by log(1+x)
Feature extraction
Interaction Features

$$(X_1, X_2) \rightarrow (1, X_1, X_2, X_1^2, X_1X_2, X_2^2)$$

```python
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])

>>> poly = PolynomialFeatures(degree=2, interaction_only=False, include_bias=True)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

Polynomial features with scikit-learn
One-Hot Encoding (OHE)

- Transform a categorical feature with \( m \) possible values into \( m \) binary features.
- If the variable cannot be multiple categories at once, then only one bit in the group can be on.

<table>
<thead>
<tr>
<th>platform</th>
<th>platform=desktop</th>
<th>platform=mobile</th>
<th>platform=tablet</th>
</tr>
</thead>
<tbody>
<tr>
<td>desktop</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mobile</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tablet</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Sparse format is memory-friendly
- sklearn.feature_extraction.DictVectorizer
Large Categorical Variables

- Common in applications like targeted advertising and fraud detection
- Example:

<table>
<thead>
<tr>
<th>categorical_feature</th>
<th>unique_values</th>
</tr>
</thead>
<tbody>
<tr>
<td>landing_page_document_id</td>
<td>636482</td>
</tr>
<tr>
<td>ad_id</td>
<td>418295</td>
</tr>
<tr>
<td>ad_document_id</td>
<td>143856</td>
</tr>
<tr>
<td>content_entities</td>
<td>52439</td>
</tr>
<tr>
<td>advertiser</td>
<td>2052</td>
</tr>
<tr>
<td>publisher</td>
<td>830</td>
</tr>
<tr>
<td>country_state</td>
<td>1892</td>
</tr>
</tbody>
</table>

Some large categorical features from Outbrain Click Prediction competition
# Feature hashing

- Hashes categorical values into vectors with fixed-length.
- Lower sparsity and higher compression compared to OHE
- Deals with new and rare categorical values (e.g., new user-agents)
- May introduce collisions

<table>
<thead>
<tr>
<th>country</th>
<th>country_hashed_1</th>
<th>country_hashed_2</th>
<th>country_hashed_3</th>
<th>country_hashed_4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>brazil</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>chile</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>venezuela</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>colombia</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>... 222 countries</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

100 hashed columns

Bin-counting

- Instead of using the actual categorical value, use a global statistic of this category on historical data.
- Useful for both linear and non-linear algorithms
- May give collisions (same encoding for different categories)
- Be careful about leakage
- Strategies
  - Count
  - Average CTR
Bin-counting

<table>
<thead>
<tr>
<th>ad_id</th>
<th>ad_views</th>
<th>ad_clicks</th>
<th>ad_CTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>423654</td>
<td>18339</td>
<td>1355</td>
<td>0.074</td>
</tr>
<tr>
<td>123646</td>
<td>335</td>
<td>12</td>
<td>0.036</td>
</tr>
<tr>
<td>68855</td>
<td>1244</td>
<td>132</td>
<td>0.106</td>
</tr>
<tr>
<td>54345</td>
<td>35387</td>
<td>1244</td>
<td>0.035</td>
</tr>
</tbody>
</table>

Counts

Click-Through Rate

\[ P(\text{click} | \text{ad}) = \frac{\text{ad}_\text{clicks}}{\text{ad}_\text{views}} \]
Textual features

- sklearn.feature_extraction.text
- Count, Tf-IDF,

Image features

- sklearn.feature_extraction.image
- Patches, img to graph,…
Feature selection
Feature selection

- Decrease complexity of model
- Remove redundant and irrelevant features
  

- `sklearn.feature_selection.SelectKBest`
  - Scoring function
  - K features
  - Threshold
  - …
Data exploration (Exploratory data analysis, EDA)

- Get the knowledge about your data
  - What does the data model look like?
  - What is the features distribution?
  - What are the features with missing or inconsistent values?
  - What are the most predictive features?
Exploratory Data Analysis (EDA)

- Get the knowledge about your data
  - get a basic description of the data
  - visualize it
  - identify patterns in it
  - identify challenges of using the data
  - form hypothesis about the data
  - …
Exploratory Data Analysis (EDA)

Basic knowledge
- Describe data (quantiles, med, mean, max, min…)
  - `head()`, `tail()`
  - `Pd.Query()`, `sorting`

Challenges
- NaNs, outliers, numeric -> nominal and vice versa

Patterns
- correlation, covariation, linear dependence, clusters, dimensionality reduction…

Visualization
- boxplots, histograms, cumulative distribution, pairplots, scatter plots, corr matrices, …
Patterns

- Pairwise correlation for dataframe columns

- Linear regression (Estimator class, fit and predict methods)

- Several variants of clustering, perhaps hierarchical one would be most appropriate

- Linear dimensionality reduction (are my features correlated?)
Visualization

Matplotlib
- Basics

Seaborn
- Extends over matplotlib (compatible calls)
- Easier processing
- Nicer graphs by default
- Several additional graph types (pairplot, facetedGrid) especially useful
- [https://www.slideshare.net/nishantupadhyaysbi/python-seaborn-cheatsheet](https://www.slideshare.net/nishantupadhyaysbi/python-seaborn-cheatsheet)

Matplotlib slides follows
More reading:
[https://elitedatascience.com/python-seaborn-tutorial](https://elitedatascience.com/python-seaborn-tutorial)
[https://seaborn.pydata.org/examples/index.html](https://seaborn.pydata.org/examples/index.html)
Supervised ML algorithms
(just the very basics now, more later)
Classification

Given labeled input data (with two or more labels), fit a function that can determine for any input, what the label is.
Regression

Given continuous input data fit a function that is able to predict the continuous value of input given other data.
K-nearest neighbors
Linear regression
Exploratory Data Analysis (EDA) - pairplot

```python
import pandas as pd
import seaborn as sns  # statistical data visualization, based on matplotlib
import matplotlib.pyplot as plt

df = pd.read_csv("C:/Users/peska/Documents/uceni/2017_2018/NFRG065/cviceni/student-mat.csv", delimiter=";")

sns.pairplot(data=df,
             x_vars=['age', 'Medu', 'Fedu', 'famrel', 'freetime', 'goout', 'absences', 'G1', 'G2'],
             y_vars=['G3'])
plt.show()
```
Exploratory Data Analysis (EDA) - CDF

```python
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("C:/Users/peska/Documents/uceni/2017_2018/NPRG065/cviceni/student-mat.csv", delimiter=";")

df.G3.hist(cumulative=True, normed=True, bins=len(df.G3))

#df.absences.hist(cumulative=True, normed=True, bins=len(df.absences))
plt.show()
```
Model Fitting & Evaluation
Model Fitting Protocol

How to learn model’s hyperparameters: grid search & cross-validation

Instead of Train – Test split, you may use additional „outer“ cross-validation

Get results from all parts of the dataset

Never use any knowledge of the test set data

E.g. For mean ratings, object similarities etc
Model Fitting Protocol

Further variants:
- Monte-Carlo cross validation:
  - random splits, arbitrary size (cold start problems)

- Bootstrap validation:
  - only one split (if you have abundant data)

- Temporal splits:
  - if domain changes over time
accuracy =
true positives + true negatives / total

precision =
true positives / (true positives + false positives)

recall =
true positives / (false negatives + true positives)

F1 score =
2 * ((precision * recall) / (precision + recall))

https://en.wikipedia.org/wiki/Precision_and_recall
**ROC** (receiver-operator curve)

True Positive Rate / False Positive Rate

**AUC** (AUROC) = area under ROC
- probability that random positive example will be ranked higher than random negative
- calculate empirically from test results

**AUPR** (area under precision-recall curve)

https://en.wikipedia.org/wiki/Receiver_operating_characteristic
MSE & Coefficient of Determination

MAE / MSE / RMSE

\[ \text{MSE} = \text{np.mean}((\text{predicted}-\text{expected})^2) \]

Coefficient of determination

R\(^2\) is a predictor of “goodness of fit” and is a value \( \in [0,1] \) where 1 is perfect fit.

- the proportion of the variance in the dependent variable that is predictable from the independent variable(s)
- independent from the scale of the output feature

\[ SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2, \quad SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2 \quad R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \]
Evaluation - sklearn

sklearn.model_selection
- GridSearchCV
- train_test_split()
- LeaveOneOut (predict for each example separately)

- Split data / Learn hyperparameters / Validate metrics
Evaluation - sklearn

sklearn.metrics: metric(true, predicted/rank,...)
Classification
-classification_report()
-precision, recall, auc, f1_score, jaccard,...
-confusion matrix

Regression
-r2_score(), MAE, MSE

Clustering
Pairwise
- cosine / euklidean distance, ...
...

```python
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report

iris = datasets.load_iris()
parameters = {'kernel': ('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

clf = GridSearchCV(svc, parameters)
clf.fit(X,y)
print(clf.cv_results_['mean_test_score'])
print(clf.cv_results_['params'])
print(clf.best_params_)

y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```
Other Evaluation

How to evaluate clusters?
Visualization (but only in 2D)
Unpredictable Future

Machine learning models attempt to predict the future as new inputs come in - but human systems and processes are subject to change.

Solution: Precision/Recall tracking over time
Pipelines
Pipelines

`skslearn.pipeline.Pipeline(steps)`
- Sequentially apply *repeatable* transformations to final estimator that can be validated at every step.
- Each step (except for the last) must implement Transformer, e.g. `fit` and `transform` methods.
- Pipeline itself implements both methods of Transformer and Estimator interfaces.
```python
>>> from sklearn.preprocessing import PolynomialFeatures
>>> from sklearn.pipeline import make_pipeline
>>> model = make_pipeline(PolynomialFeatures(2), linear_model.Ridge())
>>> model.fit(X_train, y_train)

>>> mean_squared_error(y_test, model.predict(X_test))
3.1498887586451594

>>> model.score(X_test, y_test)
0.97090576345108104
```

Pipelined Model