강의자료 링크: http://bit.ly/hgu_mlcamp_2019summer

머신러닝을 활용한 리얼월드 데이터 분석

Machine Learning with Real World Data

홍참길

charmgil@handong.edu

한동대학교 HANDONG GLOBAL UNIVERSITY

July 18, 2019 @ 전산전자공학부 여름 학기 전공 캠프

Camp Outline

	Day 1	Day 2	Day 3
13:00	Introduction Getting Started with Python	Decision Trees	Logistic Regression
14:00	Hands-on Session	Hands-on Session	Hands-on Session
15:00	*k*-Nearest Neighbors	Visualization using Matplotlib	Data Analytics Practicum
16:00	Hands-on Session	Hands-on Session	

Camp Outline

- Camp objectives
  - To learn how to handle real world data and apply ML techniques on them
  - To understand some machine learning ideas and methodologies
    - *k*-Nearest Neighbors
    - Decision Trees
    - Logistic Regression
  - To obtain hands-on experiences with ML models and algorithms

- Target audiences
  - 2nd year CS/CE students who have just taken Data Structures
    - Not assuming a thick concept of math/stat knowledge
  - Requirements
    - General knowledge on data structures (with C or C++)
    - Strong motivation

IFU

Review: Python Facts

1. Python does not support pointers
2. One function can return multiple values
3. One can use an "else" clause with a "for" loop
4. *Function Argument Unpacking* is another awesome feature
5. One can easily find the index (iteration number) inside a "for" loop using 'enumerate'

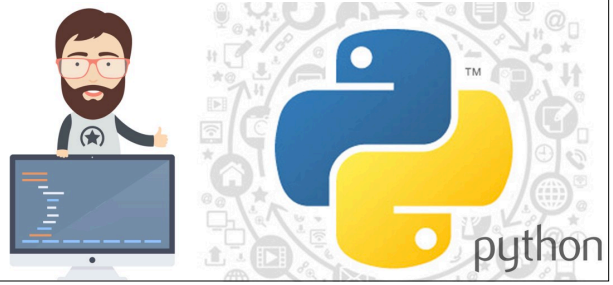


Image source: https://yourstory.com/mystory/interesting-facts-about-python-language

Review: Python Facts

6. One can chain comparison operators; e.g., 1<x<10
7. Python can define Infinities; 'Inf', '-Inf'
8. One can build a list concisely with a *list comprehension*
9. Slice operators are a way to get items from lists, as well as change them
10. There is a poem written by Tim Peters named as THE ZEN OF PYTHON which can be read by just writing `import this`

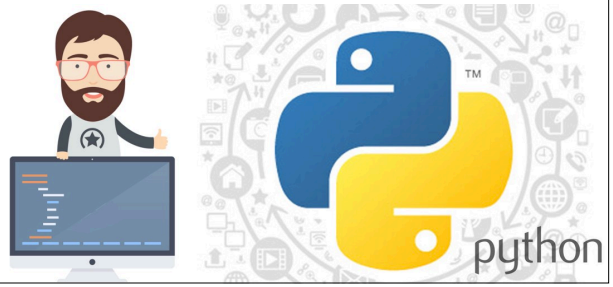
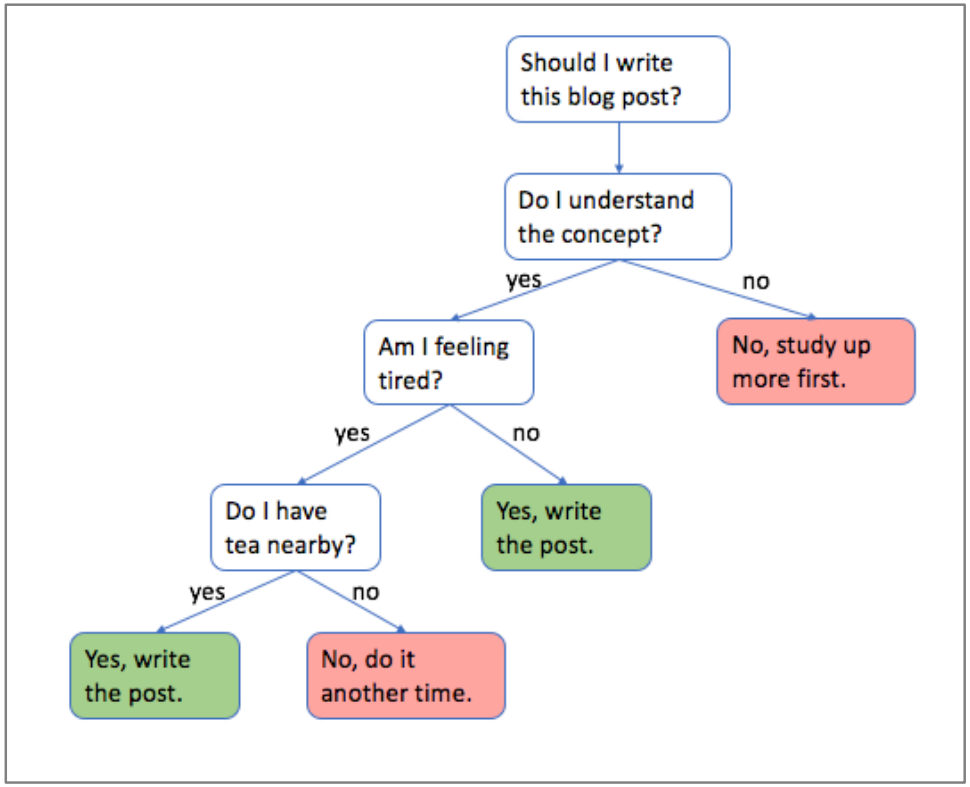


Image source: https://yourstory.com/mystory/interesting-facts-about-python-language

Review: Decision Trees (DT)

- An effective tools to represent decision rules
  - *E.g., Do I want to write a blog post about an issue?*
    Am I eligible for military service?



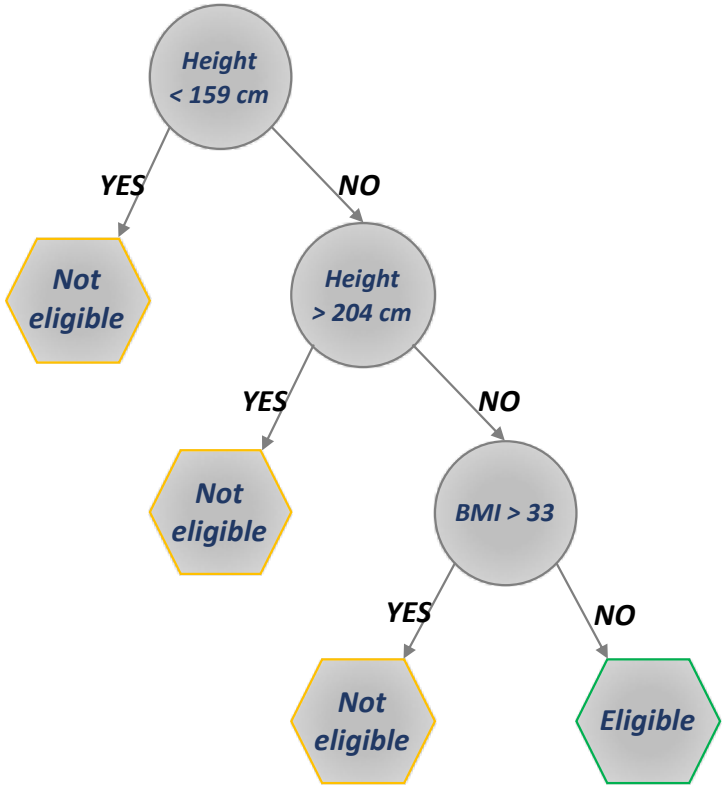


Image source: https://www.jeremyjordan.me/decision-trees/

Review: Prediction

- A decision tree $f: \mathbf{X} \rightarrow Y$
  - Assuming that a trained decision tree is given:
    - For each data instance, answer to the query on each node and take the branch with answer (move on to one of its child nodes)
    - When the instance arrives at a leaf node, make a prediction according to the node's decision
  - This could be efficiently implemented as a recursive program

Review: Training

- Idea
  - Start from an empty decision tree, with all train data
  - Split on next best attribute (feature)
    - Use, for example, information gain to decide the splitting condition

- This could be efficiently implemented as a recursive program

- Question: How to make splits?
  How to split the data into multiple nodes?

Review: Training

- Background: Entropy (degree of complexity in data)
  - High entropy
    - Y is from a uniform-like distribution
    - Near-flat histogram
    - Less predictable
  - Low entropy
    - Y is from a varied (peaks and valleys) distribution
    - Histogram has many lows and highs
    - More predictable

Review: Training

- Splitting criterion: Information gain
  - Decrease in entropy (uncertainty) after splitting a node
    - IG(X) = H(Y) - H(Y|X)

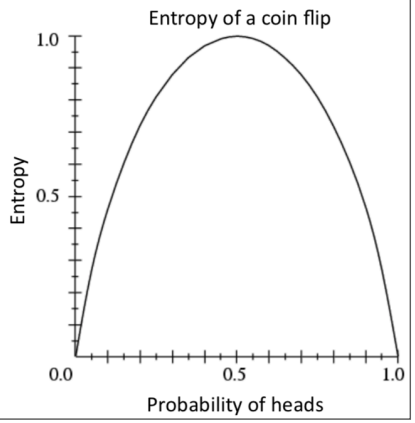
      = (Entropy before split) - (Entropy after split)

      - H(Y): Entropy of a variable Y
      - H(Y|X): Conditional entropy of Y given X

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

More uncertainty, more entropy!

Information Theory interpretation: *H(Y)* is the expected number of bits needed to encode a randomly drawn value of Y (under most efficient code)



Review: Decision Trees (DT)

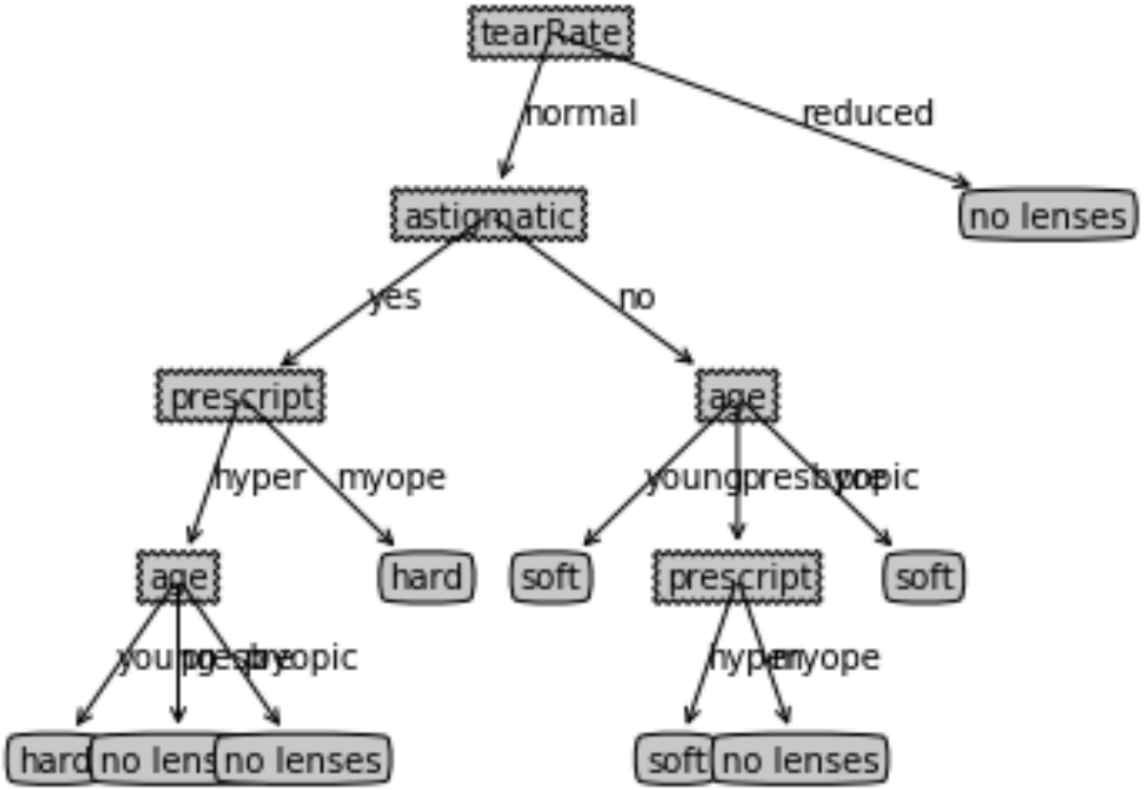
- Pros
  - Simple to understand, interpret, and visualize
  - Little effort is required for data preparation
    - Normalization is not required
  - Non-linear decision boundary could be learned
  - Can handle both numerical and categorical data

- Cons
  - Overfitting
  - High variance (model tends to be less stable)

- QnAs

Review: Visualizing Decision Trees

- Visualize the trees that we trained from data



Review: Data Visualization with Python

- Matplotlib
  - 2D plotting library which produces publication quality figures
  - One can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, *etc*.
  - Gallery (with code): https://matplotlib.org/3.1.0/gallery/index.html

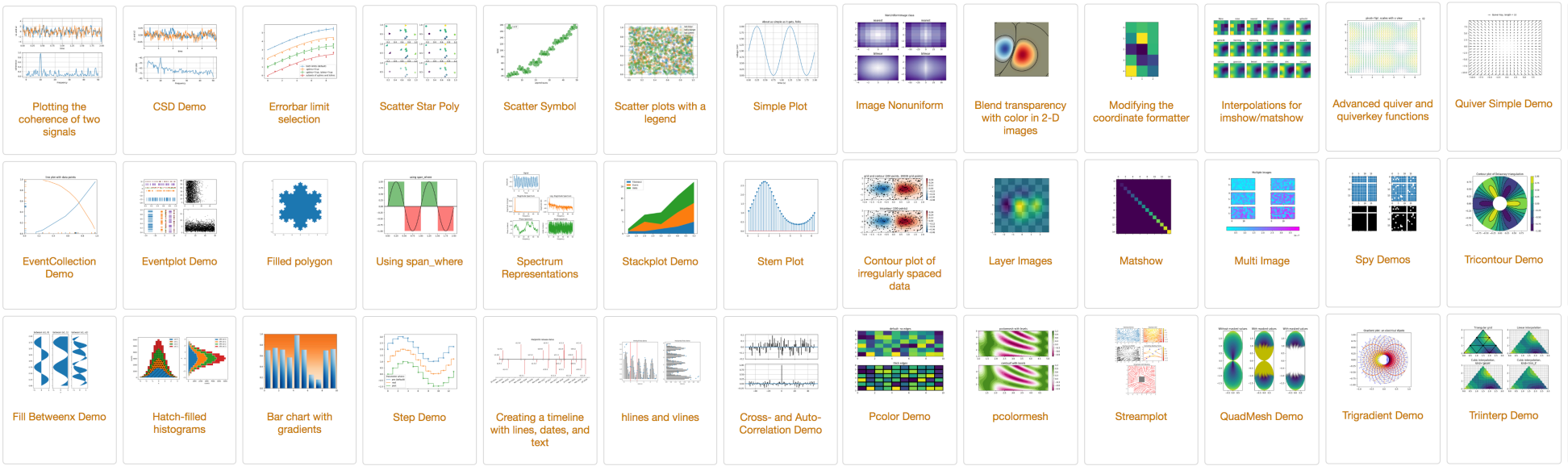


Image source: https://matplotlib.org/

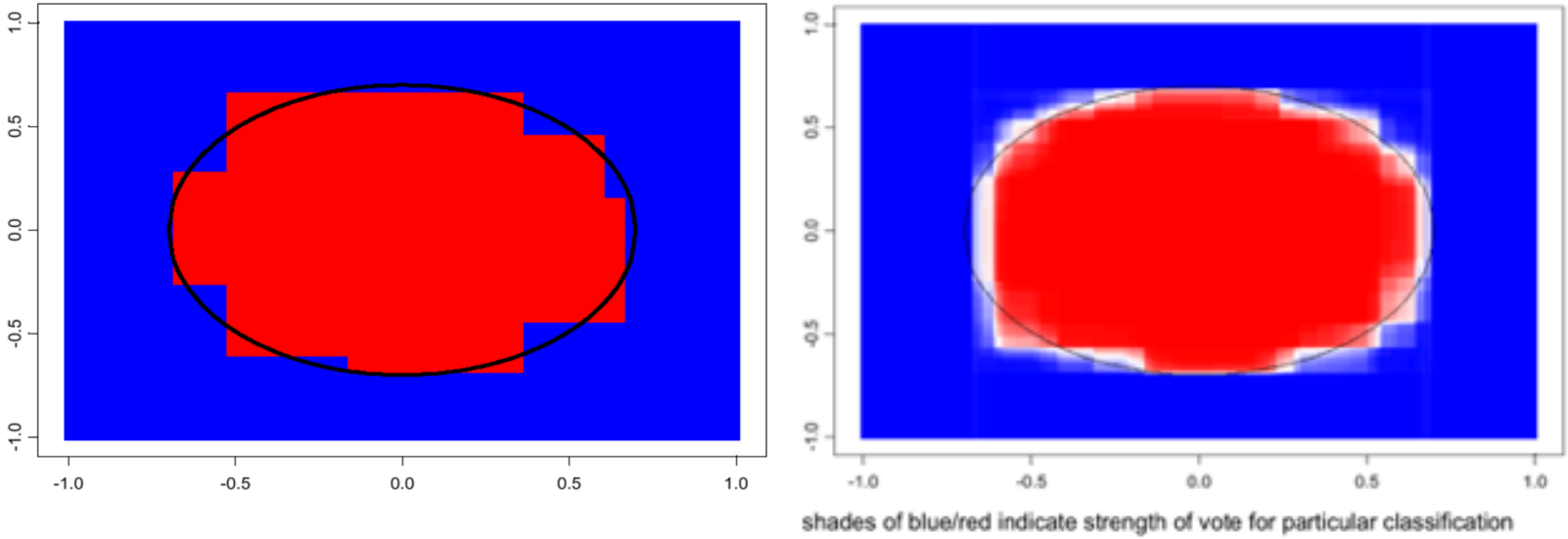
Review: Random Forests (RF)

- Issues with Decision Trees
  - Overfitting: occurs when the algorithm captures noise in the data
  - Unstable: the model can get unstable with merely small variations in data (the model could get too much sensitive)

- Solutions
  - Must use tricks to find "simple trees"
    - Early stopping of training (learning)
      - Fixed depth: do not grow trees further than a specified depth
      - Minimum population per leaf node: do not split a node if the number of data instances fall in the node is smaller than a specified number
    - Readjusting/simplifying trained trees
      - Pruning: simplify trees by merging leaf nodes

Review: Random Forests (RF)

- Ensemble (mixture) of decision tress
  - Use multiple decision trees together and improve the predictive performance
  - Illustration: using 1 tree vs. 100 trees



A/S - Decision Tree Splitting Criteria

- For classification (target variable is discrete)
  - Information Gain

$$IG(T,a) = -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{J} -\Pr(i|a) \log_2 \Pr(i|a)$$

  - Gini impurity index

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i(1-p_i) = \sum_{i=1}^{J} (p_i - {p_i}^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} {p_i}^2 = 1 - \sum_{i=1}^{J} {p_i}^2$$

- For regression (target variable is continuous)
  - Variance reduction

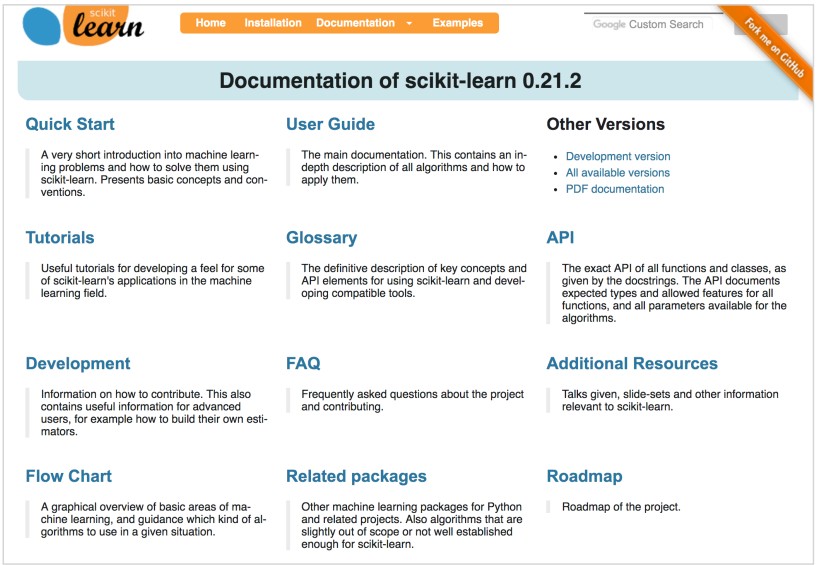
$$I_V(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2}(x_i - x_j)^2 - \left(\frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2}(x_i - x_j)^2 + \frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2}(x_i - x_j)^2 \right)$$

Agenda

- Scikit-Learn
  - Random Forests
- Logistic Regression
- Work with Real Data: NYC Taxi Data

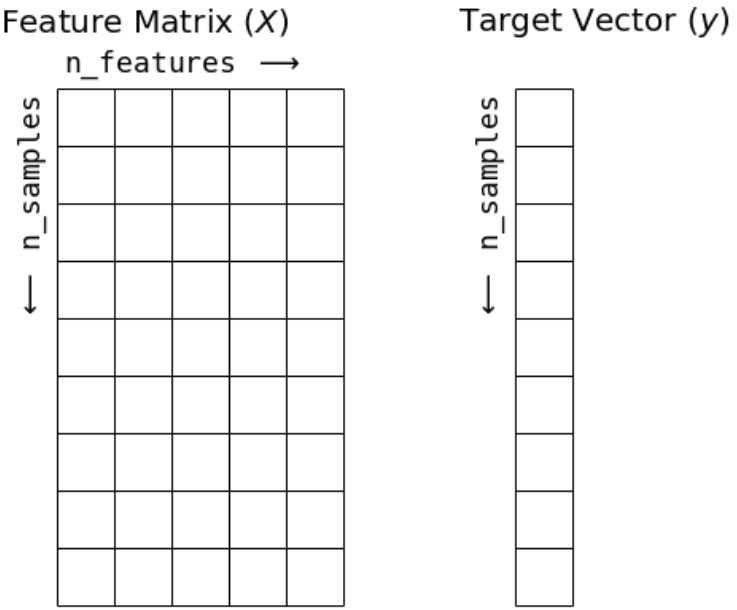
Scikit-Learn (sklearn)

- Community-driven ML library written in Python
- Simple and efficient, for both experts and non-experts
- Includes classical, well-established ML algorithms
- Built on top of NumPy and SciPy
- Shipped with documents and examples
  - https://scikit-learn.org/stable/documentation.html



Scikit-Learn (sklearn)

- Data representation
  - Most commonly used data formats are a NumPy `ndarray` or a Pandas `DataFrame/Series`
    - Each row of these matrices corresponds to one instance of the dataset
    - Each column represents a quantitative piece of information that is used to describe each instance (called "features")



Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - `sklearn.datasets`

```
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
# Only include first two training features (sepal length and sepal width)
X = X[:, :2]

print(f'First 5 samples in X: \n{X[:5]}')
print(f'Labels: \n{y}')
```

```
First 5 samples in X:
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]]
Labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree, sklearn.neighbors, ...`)

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingRegressor
from sklearn.svm import SVC, SVR
from sklearn.linear_model import LinearRegression, LogisticRegression
```

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree, sklearn.neighbors, ...`)

```
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingRegressor
from sklearn.svm import SVC, SVR
from sklearn.linear_model import LinearRegression, LogisticRegression
```

```
model = KNeighborsClassifier(n_neighbors=5)
print(model)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

Scikit-Learn (sklearn)

`sklearn.neighbors`.KNeighborsClassifier

class `sklearn.neighbors.` **`KNeighborsClassifier`**(*n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, optional (default = 5)*

Number of neighbors to use by default for `kneighbors` queries.

weights : *str or callable, optional (default = 'uniform')*

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, optional*

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree, sklearn.neighbors, ...`)

```
class Estimator(BaseClass):

    def __init__(self, **hyperparameters):
        # Setup Estimator here

    def fit(self, X, y):
        # Implement algorithm here

        return self

    def predict(self, X):
        # Get predicted target from trained model
        # Note: fit must be called before predict

        return y_pred
```

IFU

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree, sklearn.neighbors, ...`)

```
# Create the model
model = KNeighborsClassifier(n_neighbors=5)

# Fit the model
model.fit(X, y)

# Get model predictions
y_pred = model.predict(X)
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1])
```

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree, sklearn.neighbors, …`)
    - Evaluation metrics (`sklearn.metrics`)

```
# Classification metrics
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score, f1_score, log_loss)
# Regression metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
y_pred = [0, 2, 1, 3, 1]
y_true = [0, 1, 1, 3, 2]
```

```
accuracy_score(y_true, y_pred)
```

```
0.6
```

```
mean_squared_error(y_true, y_pred)
```

```
0.4
```

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree`, `sklearn.neighbors`, …)
    - Evaluation metrics (`sklearn.metrics`)
    - Experimental settings (`sklearn.model_selection`)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=2)

print(f'X.shape = {X.shape}')
print(f'X_test.shape = {X_test.shape}')
print(f'X_train.shape = {X_train.shape}')
```

Scikit-Learn (sklearn)

- API
  - https://scikit-learn.org/stable/modules/classes.html
  - Key classes
    - Datasets (`sklearn.datasets`)
    - Models (`sklearn.tree, sklearn.neighbors, …`)
    - Evaluation metrics (`sklearn.metrics`)
    - Experimental settings (`sklearn.model_selection`)

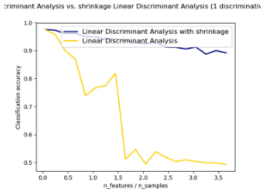
```
from sklearn.model_selection import cross_validate

clf = DecisionTreeClassifier(max_depth=2)
scores = cross_validate(clf, X_train, y_train,
                        scoring='accuracy', cv=10,
                        return_train_score=True)
```

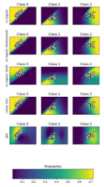
Scikit-Learn (sklearn)

- Available Models (`sklearn.tree`, `sklearn.neighbors`, …)
  - For supervised learning
    - Linear models (logistic regression)
    - Support vector machines
    - Tree-based methods (decision trees, random forests)
    - Nearest neighbors
    - Neural networks
    - Gaussian process
    - Feature selection
  - For unsupervised learning
    - Clustering
    - Matrix decomposition
    - Manifold learning
    - Outlier detection

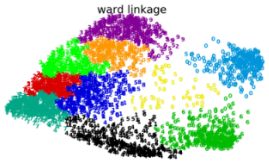
Recognizing hand-written digits



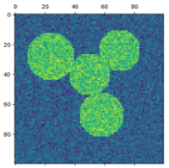
Normal and Shrinkage Linear Discriminant Analysis for classification



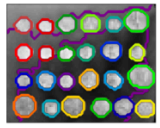
Plot classification probability



Various Agglomerative Clustering on a 2D embedding of digits



Spectral clustering for image segmentation

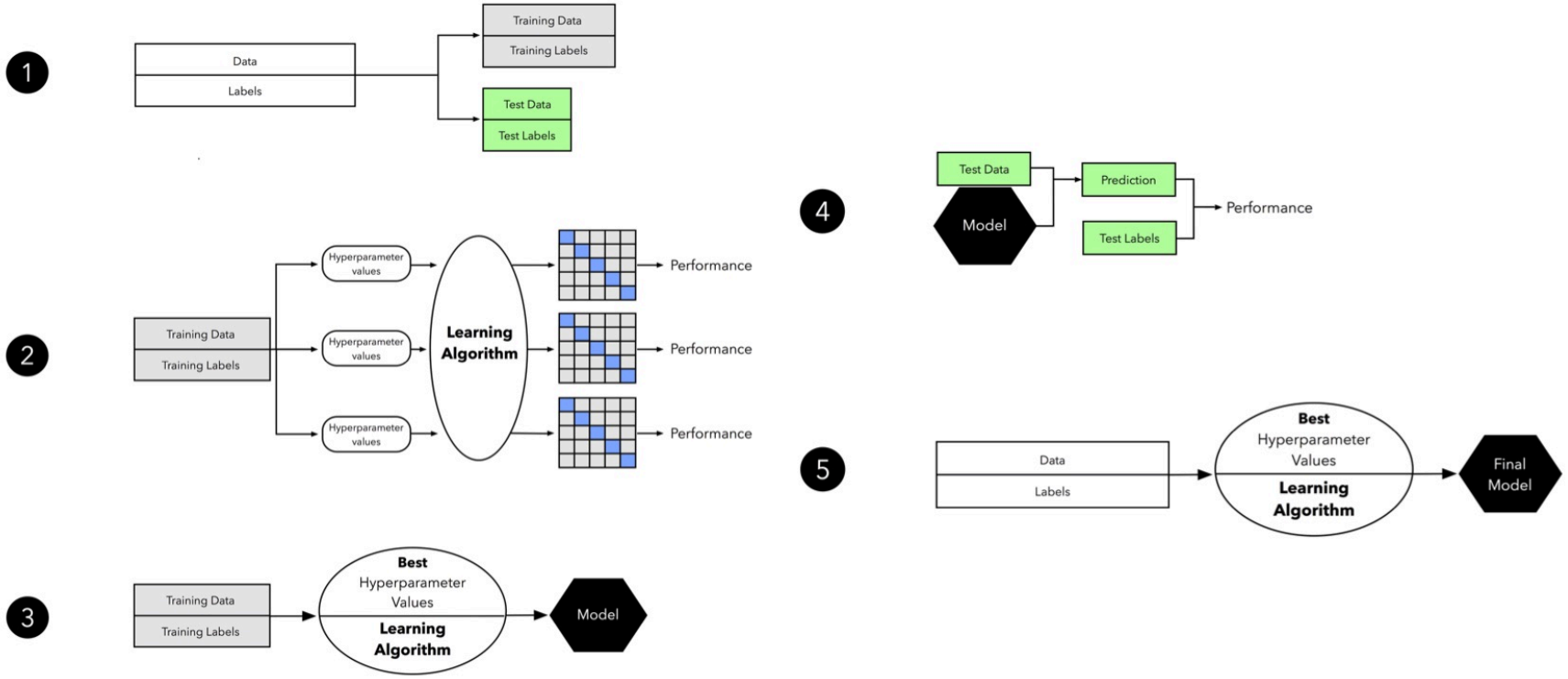


A demo of structured Ward hierarchical clustering on an image of coins

More examples: https://scikit-learn.org/stable/auto_examples/index.html

Random Forest with sklearn

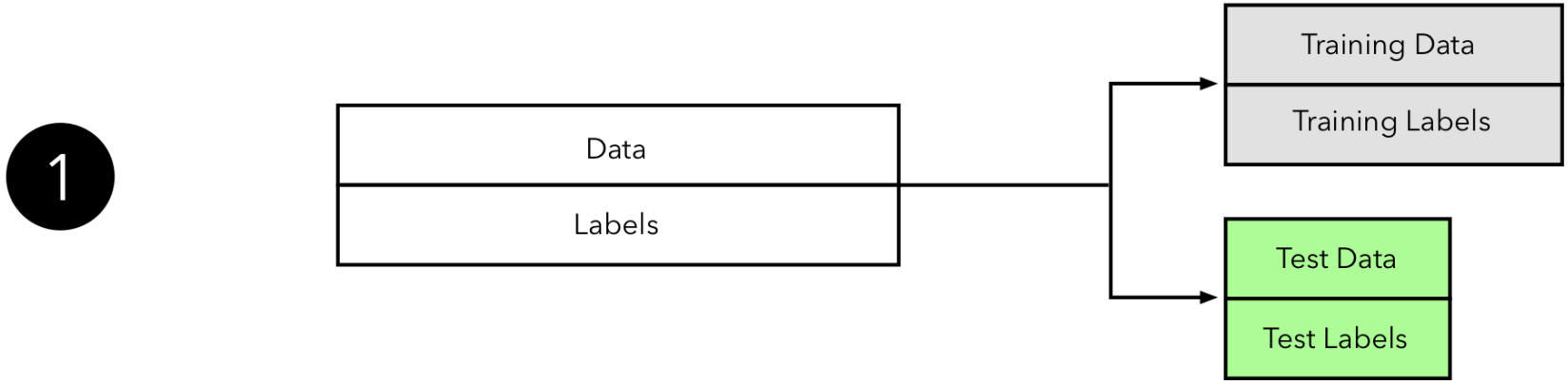
- Supervised machine learning workflow



Sources: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html
https://jrbourbeau.github.io/madpy-ml-sklearn-2018/#/0/38

Random Forest with sklearn

- Step 1 - Separating training and testing datasets

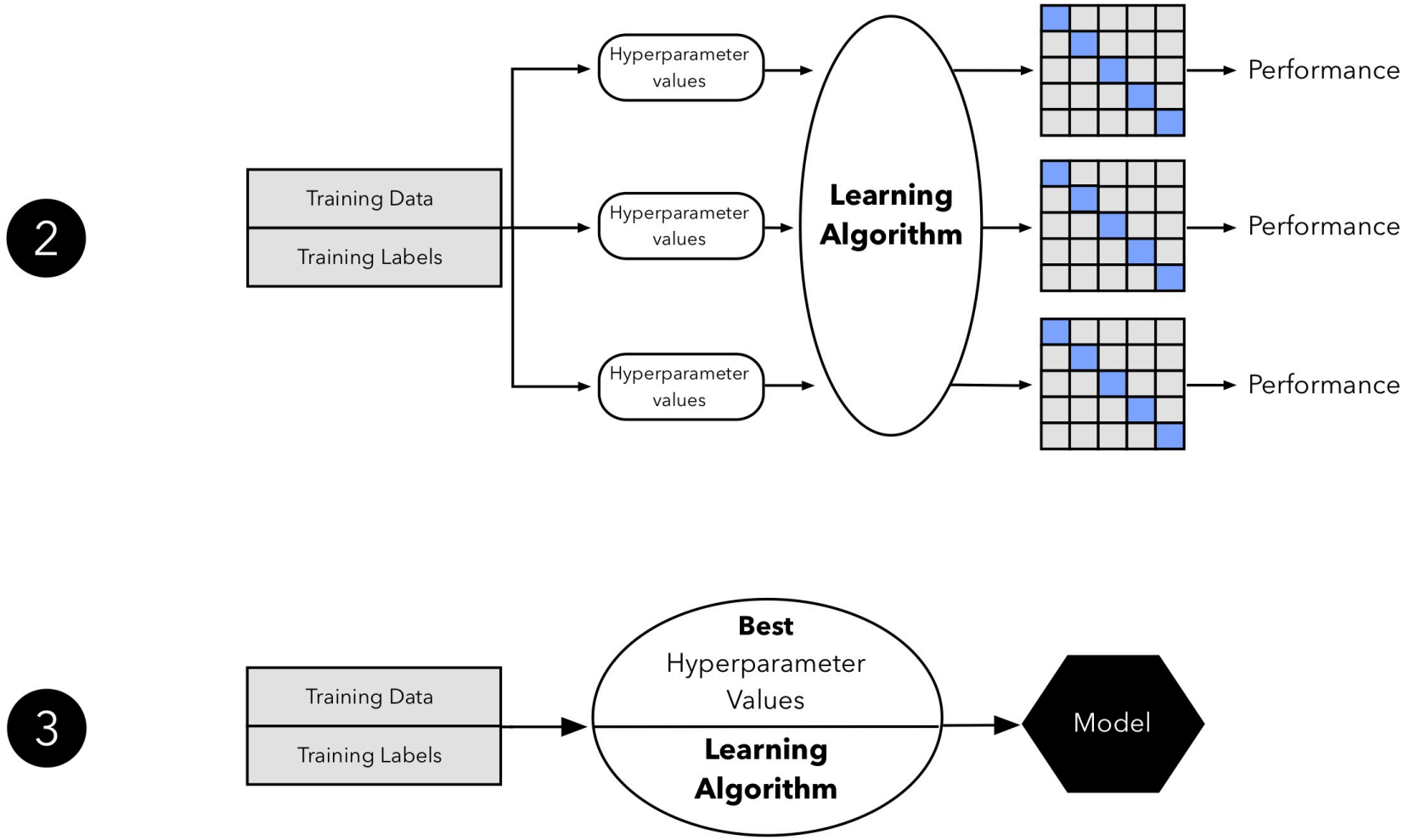



```
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3, random_state=777)
```

Sources: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html
https://jrbourbeau.github.io/madpy-ml-sklearn-2018/#/0/38

Random Forest with sklearn

- Step 2 & 3 - Optimize hyperparameters via cross validation



Sources: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html
https://jrbourbeau.github.io/madpy-ml-sklearn-2018/#/0/38

Random Forest with sklearn

- Step 2 & 3 - Optimize hyperparameters via cross validation

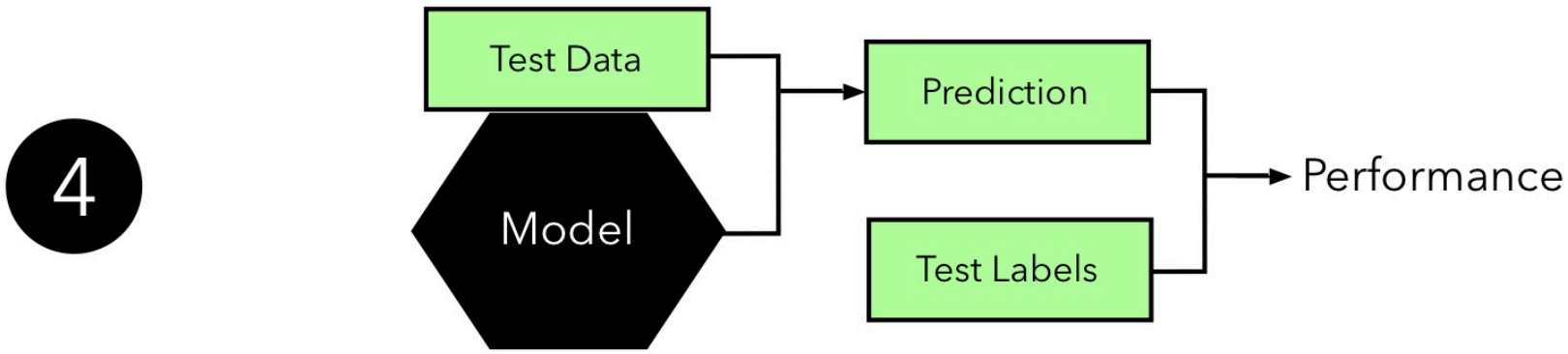
```
clf = RandomForestClassifier()
parameters = {'n_estimators': [100, 150, 200],
              'criterion': ['gini', 'entropy']}
gridsearch = GridSearchCV(clf, parameters, scoring='accuracy', cv=5)
gridsearch.fit(X_tr, y_tr)
print(f'gridsearch.best_params_ = {gridsearch.best_params_}')

best_clf = gridsearch.best_estimator_
best_clf
```

Sources: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html
https://jrbourbeau.github.io/madpy-ml-sklearn-2018/#/0/38

Random Forest with sklearn

- Step 4 - Model performance

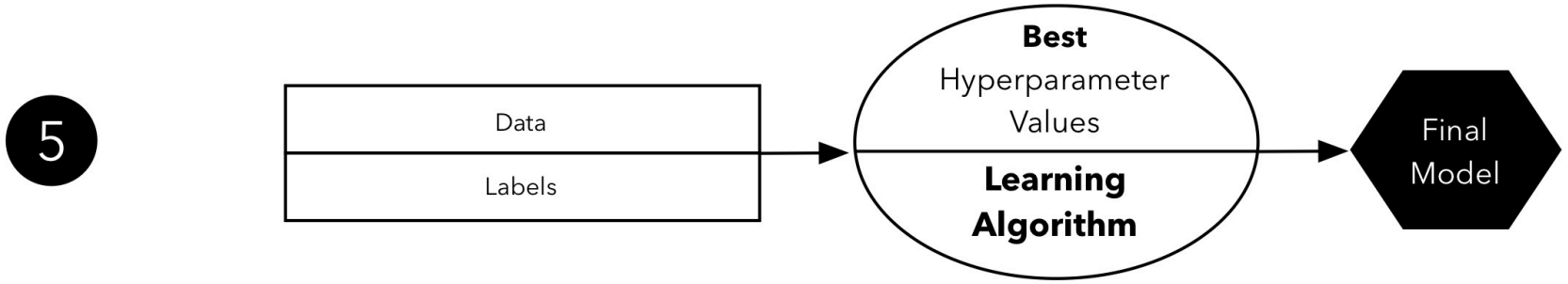



```
y_pred = best_clf.predict(X_ts)
test_acc = accuracy_score(y_ts, y_pred)
print(f'test_acc = {test_acc}')
```

Sources: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html
https://jrbourbeau.github.io/madpy-ml-sklearn-2018/#/0/38

Random Forest with sklearn

- (Optional) Step 5 - Train final model on full dataset




```
final_model = RandomForestClassifier(**gridsearch.best_params_)
final_model.fit(X, y)
```

Agenda

- Scikit-Learn
  - Random Forests
- **Logistic Regression**
- Work with Real Data: NYC Taxi Data

Logistic Regression

- Background: from linear regression to classification

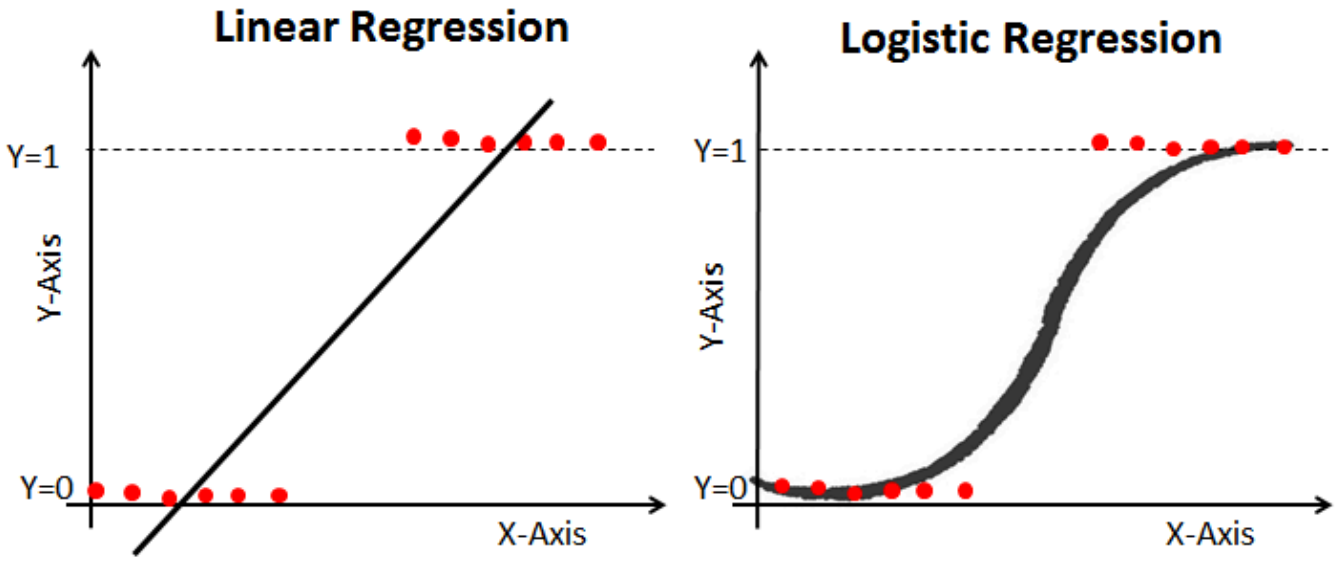


Image source: https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

Logistic Regression

- Fitting a logit function for a binary classification problem

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

$$\Leftrightarrow \frac{p}{1-p} = e^{\beta_0 + \beta_1 X}$$

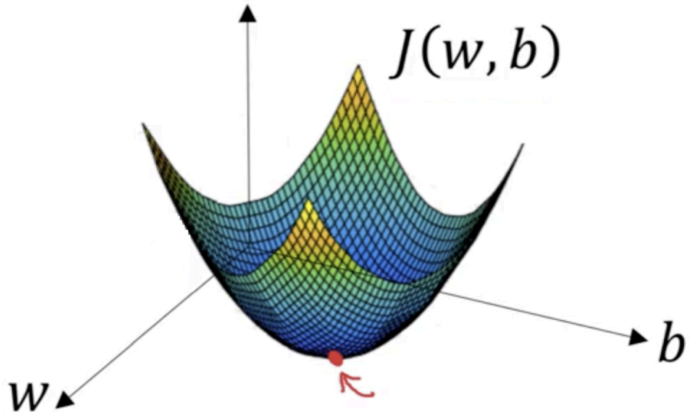
$$\Leftrightarrow p = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

- Derive a loss function: $L(\hat{y}, y) = -(y log \hat{y} + (1-y) log(1-\hat{y}))$
  - This yields a cost function:

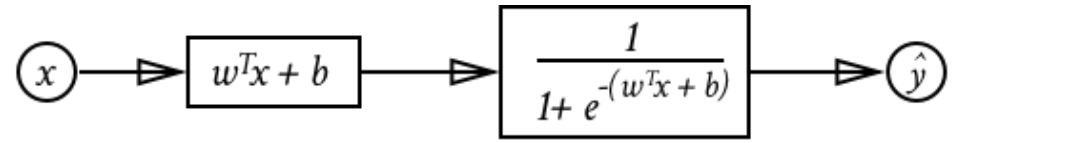
$$J(w, b) = \frac{\sum_1^m L(\hat{y}^i, y^i)}{m} = \frac{\sum_1^m y^i log \hat{y}^i + (1-y^i) log(1-\hat{y}^i)}{m}$$

Logistic Regression

- Shape of the function J(w,b): a hint for the gradient descent algorithm



- Obtained w, b form a classifier



input

linear function,
w = weight
b = bias

sigmoid function ***a***
a ∈ *(0,1)*

output based on ***a****,*
1 if a >= 0.5
0 if a < 0.5

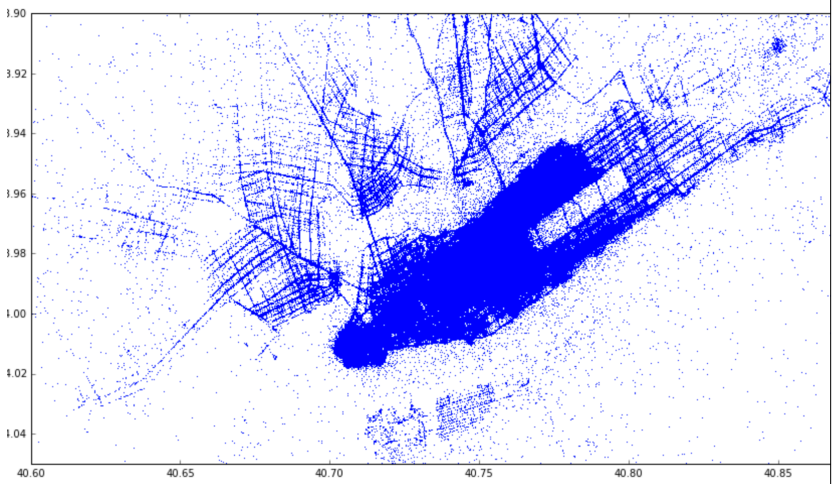
Image source: https://medium.com/technology-nineleaps/logistic-regression-gradient-descent-optimization-part-1-ed320325a67e
http://himarsh.org/the-math-of-logistic-regression/

Agenda

- Scikit-Learn
  - Random Forests
- Logistic Regression
- **Work with Real Data: NYC Taxi Data**

NYC Taxi Data

- trip_data_1.csv
  - medallion
  - hack_license
  - vendor_id
  - rate_code
  - store_and_fwd_flag
  - pickup_datetime
  - dropoff_datetime
  - passenger_count
  - trip_time_in_secs
  - trip_distance
  - pickup_longitude
  - pickup_latitude
  - dropoff_longitude
  - dropoff_latitude
- trip_fare_1.csv
  - medallion
  - hack_license
  - vendor_id
  - pickup_datetime
  - payment_type
  - fare_amount
  - surcharge
  - mta_tax
  - tip_amount
  - tolls_amount
  - total_amount



NYC Taxi Data

- Issues
  - How to merge two files into a single dataset?
  - How to find and handle outliers?
  - How to handle serial number, id, …
  - How to feed categorical features to models?
  - How to handle timestamps?

Wrap-Ups (epilogue in no specific structure)

- "Field of study that gives computers the ability to learn without being explicitly programmed" - Arthur Samuel (1959)
- "Artificial Intelligence is the new electricity" - Andrew Ng
- "By 2029, computers will have emotional intelligence and be convincing as people" - Ray Kurzweil

- Videos that were mentioned during our discussion
  - https://www.youtube.com/watch?v=pgaEE27nsQw
  - https://www.youtube.com/watch?v=hx_bgoTF7bs

- Concerns
  - Autonomous killer drones (short film) https://www.youtube.com/watch?v=DK6IGG5zRU8
  - Bill Gates: I think we do need to worry about artificial intelligence https://www.youtube.com/watch?v=EmfrMKLwr3k

IFU

Suggested Readings

- Hastie, Tibshirani and Friedman. *The Elements of Statistical Learning*. Springer, 2001.
- Murphy. *Machine Learning: a Probabilistic Perspective*. MIT Press, 2012.
- Abu-Mostafa, Magdon-Ismail and Lin. *Learning from Data*. AMLBook, 2012.

References

- Peter Harrington. *Machine Learning in Action*. Manning Publications, 2012.
- John Guttag (김영섭 역). *Introduction to Computation and Programming Using Python with Application to Understanding Data*. 2nd Ed. MIT Press, 2016.
- Henrik Brink et al. *Real-world Machine Learning*. Manning Publications, 2016.
- Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

Thank you!

Machine Learning with Real World Data (Day 3)

홍참길, 한동대학교 전산전자공학부

July 18, 2019 @ 전산전자공학부 여름 학기 전공 캠프

charmgil@handong.edu

한동대학교 HANDONG GLOBAL UNIVERSITY