CS5100 Foundations of Artificial Intelligence

Module 7 Lesson 11

Introduction to Scikit-Learn – 2

Overview

- Different Classifiers
- Preprocessing Text
- Tf-Idf
- N-grams

Decision Tree Classifier, with a Train-Test split

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()


# Now with a test-train split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.33, random_state=5)
classifier = DecisionTreeClassifier(criterion='entropy')
classifier = classifier.fit(X_train, y_train)
print(classifier)

trueVals = y_test
predictedVals = classifier.predict(X_test)

print(metrics.classification_report(trueVals, predictedVals))
print(metrics.confusion_matrix(trueVals, predictedVals))
```

Naïve Bayes, with a Train-Test split

```
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()


# Now with a test-train split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.33, random_state=5)
classifier = GaussianNB()
classifier = classifier.fit(X_train, y_train)
print(classifier)

trueVals = y_test
predictedVals = classifier.predict(X_test)

print(metrics.classification_report(trueVals, predictedVals))
print(metrics.confusion_matrix(trueVals, predictedVals))
```

Support Vector Machines (SVM), with a Train-Test split

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

# Now with a test-train split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.33, random_state=5)
classifier = SVC()
classifier = classifier.fit(X_train, y_train)
print(classifier)

trueVals = y_test
predictedVals = classifier.predict(X_test)

print(metrics.classification_report(trueVals, predictedVals))
print(metrics.confusion_matrix(trueVals, predictedVals))
```

Linear Support Vector Classification (LinearSVC), with a Train-Test split

```
from sklearn import datasets
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()


# Now with a test-train split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.33, random_state=5)
classifier = LinearSVC()
classifier = classifier.fit(X_train, y_train)
print(classifier)

trueVals = y_test
predictedVals = classifier.predict(X_test)

print(metrics.classification_report(trueVals, predictedVals))
print(metrics.confusion_matrix(trueVals, predictedVals))
```

Preprocessing Text
Cleaning up Text

1. Remove punctuation, newlines, tabs…
2. Deal with quotes, apostrophes…
3. lowercase/UPPERCASE
4. Deal with numbers, SSNs, telephone numbers
5. Deal with URLs,…
6. Code issues

CountVectorizer

Tokenize documents

Build a vocabulary of seen words

Encode new documents with that vocabulary

- For a text of N words, returns vector of length N, plus the count for each word
- Returned as scipy sparse array, can be further transformed into numpy arrays

By default, punctuation ignored, lowercased

CountVectorizer Code ..1

```
from sklearn.feature_extraction.text import CountVectorizer
text = [“…”, “…”,…]
vectorizer = CountVectorizer()

# tokenize and build vocab
vectorizer.fit(text)
print(vectorizer.vocabulary_)

vector = vectorizer.transform(text)
print(vector.shape)
print(type(vector))
print(vector.toarray())
```

CountVectorizer Code ..2

```python
from sklearn.feature_extraction.text import CountVectorizer
```

```python
text = ["R. Alexander 'Sandy' McCall Smith, CBE, FRSE (born 24 August 1948),",
"is a British-Zimbabwean writer and Emeritus Professor of Medical Law ",
"at the University of Edinburgh. In the late 20th century, McCall Smith ",
"became a respected expert on medical law and bioethics and served on ",
"British and international committees concerned with these issues. ",

"He has since become internationally known as a writer of fiction, ",
"with sales of English-language versions exceeding 40 million by 2010 ",
"and translations into 46 languages. He is most widely known as the creator of ",
"The No. 1 Ladies' Detective Agency series. ",
"'McCall' is not a middle name: his two-part surname is 'McCall Smith'." ]
```

```python
vectorizer = CountVectorizer()
# tokenize and build vocab
vectorizer.fit(text)

print(vectorizer.vocabulary_)
```

```
{'alexander': 7, 'sandy': 60, 'mccall': 46, 'smith': 64, 'cbe': 18, 'frse': 30, 'born': 15, '24': 3, 'august': 11, '1
948': 0, 'is': 38, 'british': 16, 'zimbabwean': 75, 'writer': 74, 'and': 8, 'emeritus': 25, 'professor': 57, 'of': 5
4, 'medical': 47, 'law': 45, 'at': 10, 'the': 66, 'university': 70, 'edinburgh': 24, 'in': 34, 'late': 44, '20th': 2,
'century': 19, 'became': 12, 'respected': 58, 'expert': 28, 'on': 55, 'bioethics': 14, 'served': 62, 'international':
35, 'committees': 20, 'concerned': 21, 'with': 73, 'these': 67, 'issues': 39, 'he': 32, 'has': 31, 'since': 63, 'beco
me': 13, 'internationally': 36, 'known': 40, 'as': 9, 'fiction': 29, 'sales': 59, 'english': 26, 'language': 42, 'ver
sions': 71, 'exceeding': 27, '40': 4, 'million': 49, 'by': 17, '2010': 1, 'translations': 68, 'into': 37, '46': 5, 'l
anguages': 43, 'most': 50, 'widely': 72, 'creator': 22, 'no': 52, 'ladies': 41, 'detective': 23, 'agency': 6, 'serie
s': 61, 'not': 53, 'middle': 48, 'name': 51, 'his': 33, 'two': 69, 'part': 56, 'surname': 65}
```

CountVectorizer Code ..3

```python
print(vectorizer)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

```python
vector = vectorizer.transform(text)

print(vector.shape)

print(vector.toarray())
```

```
(10, 76)
[[1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 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 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
  0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  1 1]
 [0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
  0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 2 0 0 0 1 0 0 0
  0 0]
 [0 0 0 0 0 0 0 0 2 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 2 0 0 1 0 0 0 1 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 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
  0 0 1 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 1
  0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1
  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
  1 0]
 [0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1
  0 0]
 [0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
```

Term frequency tf

Term frequency $\mathrm{tf}_{t,d}$: number of times term *t* occurs in document *d*.

Can we use tf when representing the document d?

Raw term frequency (count) not what we want:

- Document with 10 occurrences of a term
  - more about the term than a document with only 1 occurrence
  - but not 10 times more about that term

Log-frequency weighting

The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

[Why log, why 1 + ?]

Document frequency ..1

Rare terms more informative than frequent terms

- E.g. stop words

Consider a term that is rare in the collection
(e.g., *defenestration*)

A document containing this term is very likely to be relevant to the topic of *defenestration*

→ We want a high weight for rare terms like

defenestration

Document frequency ..2

Frequent terms less informative than rare terms

Consider a term that is frequent in a collection of documents (e.g., *high, increase, line*)

- A document containing such a term is more likely to be about that term than a document that doesn't
- But it's not a sure indicator

For frequent terms,

- we want large positive weights for words like *high, increase, and line*
- but lower weights than for rare terms.

Document frequency (df) used to capture this

idf weight

df_t is the document frequency of t: the number of documents that contain t

- df_t is an inverse measure of the 'informativeness' of t
- $\text{df}_t \leq N$, *the total number of documents*

idf (inverse document frequency) of t defined by

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

- log (N/df_t) used instead of N/df_t to "dampen" the effect of idf
- base of log will not matter

Tf * idf

Product of TF and IDF

- Picks relevant words, weighs them appropriately
- Frequency and Rarity both given weight

Very popular, especially in information retrieval applications

TfidfVectorizer

```
from sklearn.feature_extraction.text
        import TfidfVectorizer
text = ['…','…',…]

vectorizer = TfidfVectorizer()

vectorizer.fit(text)

print(vectorizer.vocabulary_)
print(vectorizer.idf_)

vector = vectorizer.transform([text[0]])

print(vector.shape)
print(vector.toarray())
```

```
print(vectorizer)
print([text[0]])
```

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
["R. Alexander 'Sandy' McCall Smith, CBE, FRSE (born 24 August 1948),"]
```

```
vector = vectorizer.transform([text[0]])

print(vector.shape)

print(vector.toarray())
```

```
(1, 76)
[[ 0.33138268  0.          0.          0.33138268  0.          0.          0.
   0.33138268  0.          0.          0.          0.33138268  0.          0.
   0.          0.33138268  0.          0.          0.33138268  0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.33138268  0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.24645907  0.          0.
   0.          0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.33138268  0.          0.
   0.          0.24645907  0.          0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.        ]]
```

Ngrams

Ngrams: contiguous sequence of *n* items

Character n-grams

Input: Artificial

Unigrams: A, r, t, i, f …

Bigrams: Ar, rt, ti, if, fi …

Trigrams: Art, rti, …

4-grams: Arti, rtif, … and so on

Word n-grams

Input: The quick brown fox jumped over the lazy dog

Unigrams: The, quick, brown, …

Bigrams: The quick, quick brown, brown fox, …

Trigrams: The quick brown, quick brown fox, brown fox jumped, …

4-grams: The quick brown fox, quick brown fox jumped, … and so on

Additional features

lowercase, analyzer, stop_words, ngram_range, max_df and min_df, max_features

```
class sklearn.feature_extraction.text.TfidfVectorizer(
input='content', encoding='utf-8',  decode_error='strict',
strip_accents=None, lowercase=True,
preprocessor=None, tokenizer=None, analyzer='word',
stop_words=None,
token_pattern='(?u)\b\w\w+\b',
ngram_range=(1, 1),
max_df=1.0, min_df=1,
max_features=None,
vocabulary=None, binary=False,
dtype=<class 'numpy.int64'>, norm='l2',
use_idf=True, smooth_idf=True,
sublinear_tf=False)
```

Pipeline

```
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB()),
          ])
```

GridSearchCV

```
from sklearn.model_selection import GridSearchCV
 parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
               'tfidf__use_idf': (True, False),
               'clf__alpha': (1e-2, 1e-3) }

gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)


print gs_clf.best_score_
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs_clf.best_params_[param_name]))
```

TfidfVectorizer + FeatureUnion + Pipeline + GridSearchCV

```
classifier = SVC(probability=True)

# Create combined estimator from
# Word and Char ngram vectorizers + classifier
    vectorizerW = TfidfVectorizer(analyzer='word', lowercase=True)
    vectorizerC = TfidfVectorizer(analyzer='char', lowercase=True)
    combined_features = FeatureUnion([("word", vectorizerW), ("char", vectorizerC)])

    pipeline = Pipeline([("features", combined_features), ("classifier", classifier)])

    # by default, n_jobs = 1.

    grid_search = GridSearchCV(pipeline, param_grid=param_grid, verbose=10, n_jobs=-1)
```

Parameter Grid

```
# param_grid = dict(
    features__word__max_features=[2000, 4000],
    features__char__max_features=[2000, 4000, 8000],
    features__word__min_df=[2, 3],
    features__char__min_df=[2, 3],
    features__word__ngram_range=[(1, 2), (1, 3)],
    features__char__ngram_range=[(3, 3), (3, 4), (4, 4)],
    classifier__class_weight=['balanced', None],
    classifier__C=[1, 10],
    classifier__kernel=['linear']
)
```

Best score, best param values, prediction

```
print("Best score: %0.3f" % grid_search.best_score_)
    print("Best parameters set:")
    best_parameters = grid_search.best_estimator_.get_params()
    for param_name in sorted(param_grid.keys()):
        print("\t%s: %r" % (param_name, best_parameters[param_name]))

    sys.stdout.flush()

    # Run the grid_search transforms+prediction with best parameters on test data
    y_pred = grid_search.predict(X_test)
     …
```

Multiclass vs. Multilabel

Multiclass classification means a classification task with more than two classes; e.g., classify a set of images of fruits which may be oranges, apples, or pears. Multiclass classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time.

Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A text might be about any of religion, politics, finance or education at the same time or none of these.

http://scikit-learn.org/stable/modules/multiclass.html#multiclass

Project Details

Project document

Strawman program

Data files

Further Information

Home page: http://scikit-learn.org/stable/

Installation: http://scikit-learn.org/stable/install.html

Quick Start: http://scikit-learn.org/stable/tutorial/basic/tutorial.html

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

Examples: http://scikit-learn.org/stable/auto_examples/index.html

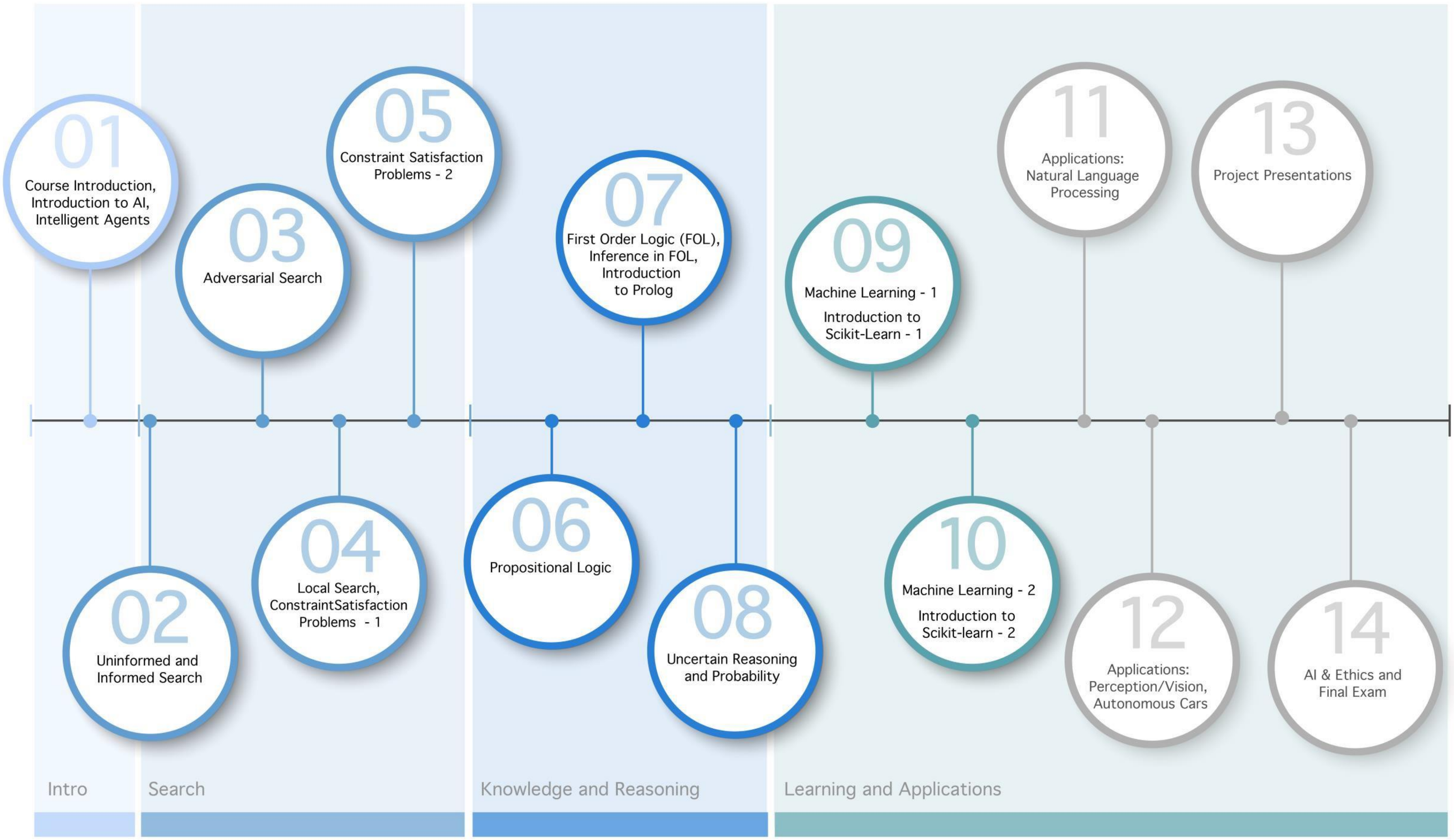
Github: https://github.com/scikit-learn

User Guide (PDF): http://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf

Plus

https://www.oreilly.com/ideas/intro-to-scikit-learn

http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction


01
Course Introduction, Introduction to AI, Intelligent Agents
02
Uninformed and Informed Search
03
Adversarial Search
04
Local Search, ConstraintSatisfaction Problems - 1
05
Constraint Satisfaction Problems - 2
06
Propositional Logic
07
First Order Logic (FOL), Inference in FOL, Introduction to Prolog
08
Uncertain Reasoning and Probability
09
Machine Learning - 1
Introduction to Scikit-Learn - 1
10
Machine Learning - 2
Introduction to Scikit-learn - 2
11
Applications: Natural Language Processing
12
Applications: Perception/Vision, Autonomous Cars
13
Project Presentations
14
AI & Ethics and Final Exam
Intro
Search
Knowledge and Reasoning
Learning and Applications