Artificial Intelligent and Its Application

Laboratory 3: Scikit-learn

Dr Patrick Chan
School of Computer Science and Engineering
South China University of Technology

Outline

- A brief introduction to Scikit-learn (sklearn)
- Data Pre-processing
- Training
- Evaluation
- Dataset Generation
- Unsupervised learning

What is Scikit-learn?

- A power library for machine learning
- User-friendly APIs
- Based on NumPy and SciPy, run fast

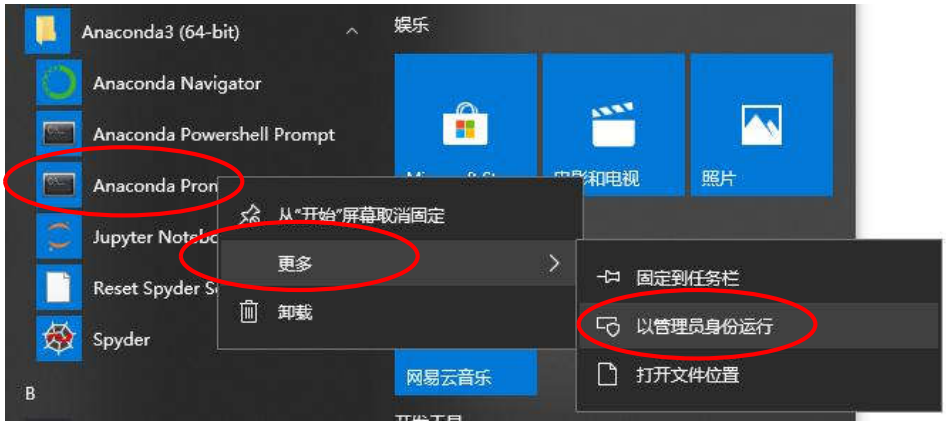
Why Scikit-learn?

- Contains a lot of ML algorithms
  - Decision tree, SVM, k-NN, MLP, etc
- Consistent APIs
  - Same standard interface for all algorithms
- Data processing
  - For example: scaling, sampling
- Integrate open source datasets
  - For example: mnist, iris

Install Scikit-learn

- Use Anaconda to install Scikit-learn
  - `conda install -c anaconda scikit-learn`

Right click



Install Scikit-learn

- Use Anaconda to install Scikit-learn
  - `conda install -c anaconda scikit-learn`

- scikit-learn will be downloaded and installed

Install Scikit-learn

- Check if scikit-learn is installed
  - Try to import **sklearn**

```
import sklearn

sklearn
```

```
<module 'sklearn' from 'C:\\Users\\xh
te-packages\\sklearn\\__init__.py'>
```


Simple Example

Data Preparation

```
import numpy as np
dataset = np.array([
    [6, 148, 0], [8, 183, 0],
    [2, 197, 0], [3, 78, 0],
    [1, 85, 1],  [5, 116, 1],
    [10, 115, 1],[1, 103, 1],
])
labels = dataset[:, 2]
data = dataset[:, :2]
```

Classifier

```
from sklearn.naive_bayes import GaussianNB
cls = GaussianNB()
```

Training

```
cls = cls.fit(data, labels)
```

Prediction

```
pred = cls.predict(data)
```

Type of Functions

1. Dataset Preparation
2. Data Preprocessing
3. Classifier
4. Training/Prediction
5. Evaluation
6. Plotting
7. Unsupervised Learning

DATASET PREPARATION

Dataset Format

- Dataset used in Scikit-learn should be in numpy array format
- Depend on the file type, different functions in libraries can be used to load the dataset into the memory
- Data stored in other format should be converted to numpy

Load Data from a file

- Example:

Dataset in npy (numpy format)

- `dataset = numpy.load(filename)`

Dataset in csv (pandas should be used)

- `df = pandas.read_csv(filename)`
- `dataset = df.values` *Convert to numpy*

Load build-in Data

Scikit-learn contains some well-known datasets in a library called datasets

- `from sklearn import datasets`

E.g. mnist dataset

- `datasets.load_digits()`
  - Both data and labels are **numpy array**

Load build-in Data

Other well-known datasets

- `datasets.load_iris()`
- `datasets.load_boston()`
- `datasets.load_wine()`
- `datasets.load_breast_cancer()`
- `datasets.load_diabetes()`

- Reference:
  https://scikit-learn.org/stable/datasets/index.html

Generation

Generate data randomly

- `make_classification([samNum, featNum, claNum])`
  - `samNum`: sample #
  - `featNum`: feature #
  - `claNum`: class #
  - return: numpy arrays: data and labels

- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#sklearn.datasets.make_classification

Generation

Generate data followed Normal distribution

- `make_classification([mean, cov, samNum, featNum, claNum])`
  - `mean`: array, mean of each dimenion
  - `cov`: matrix, the covariance matrix
  - `samNum`: sample #
  - `featNum`: feature #
  - `claNum`: class #
  - return: numpy arrays: data and labels

Generation

More functions for generating data

- `make_biclusters(`*`shape`*`,` *`clusterNum`*`)`
- `make_circles([`*`samNum`*`])`

- Reference:
  https://scikit-learn.org/stable/modules/classes.html#samples-generator

DATA PREPROCESSING

Scaling

Value Scaling

- `from sklearn.preprocessing import MinMaxScaler`
- Transform the value and the label

Scaling

- `MinMaxScaler([FectRange, copy])`
  - Normalize the features values into a range
  - `FectRange`: tuple, normalized range
  - `copy`: bool, True: the old data will be copied
- `StandardScalar()`
  - x = (x - u) / s, where u, s are mean and standard deviation
- `Normalizer()`
  - Normalize samples individually to unit norm

Scaling

- An example

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler((0, 1))
data = [
    [1, 2, 3, 4],
    [2, 5, 7, 3],
    [1, 4, 4, 5]
]
data = scaler.fit_transform(data)
data
```

min: 3
max: 5
4 -> (4-min)/(max-min) = 0.5
3 -> (3-min)/(max-min) = 0.0
5 -> (5-min)/(max-min) = 1.0

```
array([[0.        , 0.        , 0.        , 0.5       ],
       [1.        , 1.        , 1.        , 0.        ],
       [0.        , 0.66666667, 0.25      , 1.        ]])
```


Scaling

Label Encoding

- `LabelEncoder()`
  - Encode labels with value between 0 and c-1, c is the class number

- Reference:
  https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

Training and Test Set

Split dataset into training and test set

- `train_test_split(data, label[, test_size])`
  - `data`: numpy array, all data
  - `label`: all corresponding labels
  - `test_size`: float, proportion of the test set
  - return: numpy arrays: training set, test set, training labels and test labels

Training and Test Set

- Example

```
from sklearn.model_selection import train_test_split

train_set, test_set, train_label, test_label = train_test_split(
    data, labels, test_size=0.2)
```

80% for training,
20% for test

```
train_set[:3]
```

```
array([[0.05882353, 0.90954774],
       [0.        , 0.69346734],
       [0.23529412, 0.47738693]])
```

```
test_set[:3]
```

```
array([[0.        , 0.88944724],
       [0.        , 0.83919598],
       [0.23529412, 0.55276382]])
```

Feature Selection

- `VarianceThreshold([threshold])`
  - Remove feature with low variance
  - `threshold`: float, remove the features with variance not larger than threshold

2 features' variance are both 0, will be removed

```
from sklearn.feature_selection import VarianceThreshold

X = np.array([
    [0, 2, 0, 3],
    [0, 1, 4, 3],
    [0, 1, 1, 3]
])
selector = VarianceThreshold()
X = selector.fit_transform(X)
X
```

```
array([[2, 0],
       [1, 4],
       [1, 1]])
```


Feature Selection

Select k features with highest scores

- `selectKBest([score_func, k])`:
  - score_func: a function, to calculate the scores

Select % features with highest scores

- `selectPercentile([score_func, percentile])`
  - score_func: a function, to calculate the scores

- Reference:
  https://scikit-learn.org/stable/modules/feature_selection.html

Try It

- Given a dataset with 4 samples
  [[1, 2, 5, 6, 3], [1, 3, 8, 5, 5],
  [2, 5, 2, 6, 4], [1, 6, 1, 5, 4]]
  - Remove the features whose variance is <= 0.3
  - Select the best 2 feature by Chi-squared stats

Resampling

Resampling for imbalance problem

- `conda install -c conda-forge imbalanced-learn`
  - Same as installing sklearn

Oversampling

- `from imblearn.over_sampling import RandomOverSampler`
- `RandomOverSampler([sam_strategy, random_state])`
  - `sam_strategy`: specify how to resample the dataset
  - `random_state`: control randomization of algorithm
- Reference:
  https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.RandomOverSampler.html

Oversampling

```
from collections import Counter

X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.01, 0.05, 0.94],
                           class_sep=0.8, random_state=0)
sorted(Counter(y).items())
```

```
[(0, 64), (1, 262), (2, 4674)]
```

Before sampling:
Class0 64 samples
Class1 262 samples
Class2 4674 samples

```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=0)
X_resampled, y_resampled = ros.fit_resample(X, y)
sorted(Counter(y_resampled).items())
```

```
[(0, 4674), (1, 4674), (2, 4674)]
```

After sampling:
Each class 4674 samples

Undersampling

- `from imblearn.under_sampling import RandomUnderSampler`
- `RandomUnderSampler([sam_strategy, random_state])`
  - `sam_strategy`: specify how to resample the dataset
  - `random_state`: control randomization of algorithm

- Reference:
  https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html

Undersampling

```
from collections import Counter

X, y = make_classification(n_samples=5000, n_features=2, n_informative=2,
                           n_redundant=0, n_repeated=0, n_classes=3,
                           n_clusters_per_class=1,
                           weights=[0.01, 0.05, 0.94],
                           class_sep=0.8, random_state=0)
sorted(Counter(y).items())
```

```
[(0, 64), (1, 262), (2, 4674)]
```

Before sampling:
Class0 64 samples
Class1 262 samples
Class2 4674 samples

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=0)
X_resampled, y_resampled = rus.fit_resample(X, y)
sorted(Counter(y_resampled).items())
```

```
[(0, 64), (1, 64), (2, 64)]
```

After sampling:
Each class 64 samples

Resampling

Other resampling technique

- Over sampling
  - `SMOTE()`
  - `ADASYN()`
  - Reference:
    https://imbalanced-learn.org/en/stable/over_sampling.html
- Under sampling
  - `ClusterCentroids()`
  - `EditedNearestNeighbours()`
  - Reference:
    https://imbalanced-learn.org/en/stable/under_sampling.html

CLASSIFIER DEFINITION

Classifier

- Bayes Classifier
- Decision tree
- SVM
- K-NN
- MLP
- Random forest
- Ensemble

Bayes Classifier

- `from sklearn import GaussianNB`
- `cls = GaussianNB([priors, var_smoothing])`
  - `priors`: Prior probabilities of the classes. If specified the priors are not adjusted according to the data.
  - `var_smoothing`: Portion of the largest variance of all features that is added to variances for calculation stability
- Reference:
  https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes

Classifier

Decision Tree

- `from sklearn import tree`
- `cls = tree.DecisionTreeClassifier([max_depth])`
  - `max_depth`: maximum depth of the tree, no limitation by default
  - Supports multi-label classification
- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Classifier

SVM

- `from sklearn.svm import SVC`
- `cls = SVC([kernel, gamma])`
  - `kernel`: str, specify the kernel type
  - `gamma`: kernel coefficient for “rbf”, “poly” and “sigmoid”
  - Supports multi-label classification
- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Classifier

K-NN

- `from sklearn.neighbors import KNeighborsClassifier`
- `cls = KNeighborsClassifier([n_neighbors])`
  - `n_neighbors`: Number of neighbors to use for kneighbors queries, 5 by default
- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Classifier

MLP

- `from sklearn.neural_network import MLPClassifier`
- `cls = MLPClassifier([activation, ..])`
  - `activation`: str, activation function for the hidden layer, “relu” by default
  - `hidden_layer_sizes`: tuple, the ith number represents the number of neurons in the ith hidden layer
  - `batch_size`: int, decide how many samples are used for one update, min(200, n_samples) by default

MLP

- `solver`: str, the solver for weight optimization, “adam” by default
- `alpha`: float, L2 penalty (regularization term) parameter, 0.0001 by default
- `learning_rate_init`: float, initial learning rate, 0.001 by default, only used when `solver`=“sgd”
- `learning_rate`: str, learning rate schedule, “constant” by default, only used when `solver`=“sgd”
- `max_iter`: int, maximum number of iterations, 200 by default

MLP

- `shuffle`: bool, set True to shuffle samples in each iteration, True by default
- `momentum`: float, between 0 to 1, momentum for gradient update, 0.9 by default, only used when `solver`=“sgd”

- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Ensemble: Random Forest

- `from sklearn.ensemble import RandomForestClassifier`
- `cls = RandomForestClassifier([n_estimators])`
  - `n_estimators`: int, n decision trees, 10 by default
  - Supports multi-label classification
- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Ensemble: Voting

- `from sklearn.ensemble import VotingClassifier`
- `voting = VotingClassifier(estimators[, voting])`
  - `estimators`: list, contains tuples as (str, estimator)
  - `voting`: 'hard', majority voting; if 'soft', argmax of the sums of the predicted probabilities
- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html#sklearn.ensemble.VotingClassifier

Ensemble: Bagging

- `from sklearn.ensemble import BaggingClassifier`
- `bagging = BaggingClassifier([base_estimator, n_estimators, max_samples, max_features])`
  - `base_estimator`: the base estimator to fit on random subset of the dataset, decision tree by default
  - `n_estimators`: int, n estimators in the ensemble, 10 by default
  - `max_samples`: int or float, specify how many samples are drawn from the whole dataset

Ensemble: Bagging

  - `max_features`: int or float, specify how many features are drawn from all features

- Reference: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Ensemble: Boosting

- `from sklearn.ensemble import GradientBoostingClassifier`
- `boosting= GradientBoostingClassifier([n_estimators])`
  - `n_estimators`: int, the number of boosting stages to perform, 10 by default
  - Supports multi-label classification
- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Load/Save Classifier

Load a trained classifier by pickle

- `import pickle as pkl`
- `f = open(filepath, "rb")`
- `classifier = pkl.load(f)`
- `f.close()`
  - A trained classifier will be loaded from disk according to the given filepath

Load/Save Classifier

Save a trained classifier by pickle

- `import pickle as pkl`
- `f = open(filepath, "wb")`
- `pkl.dump(classifier, f)`
- `f.close()`
  - The trained classifier will be saved to the given filepath

Classifier

- Reference:
  https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Try It

- Given dataset in “lab2_TryIt.csv”
  - Train a **Bayes classifier** on the dataset, calculate the test accuracy (Split the dataset into training set and test set)
  - Evaluate its performance in terms of F1-score, average precision, ROC-curve and AUC

Try It

- Given dataset in “lab2_TryIt.csv”
  - Train a **decision tree** classifier on the dataset, calculate the test accuracy
  - Change the depth of the decision tree, calculate the test accuracy

Try It!

- Given dataset in “lab2_TryIt.csv”
  - Train a **SVM** classifier on the dataset, calculate the test accuracy
  - Use different kernels for the SVM, and compare the test accuracies

Try It!

- Given dataset in “lab2_TryIt.csv”
  - Train **k-NN** classifiers on the dataset where k from 1 to 5
  - Find out the corresponding value of k when the test accuracy is the highest

Try It!

- Given dataset in “lab2_TryIt.csv”
  - Train a **MLP with the following structures** on the dataset and calculate test accuracy
    - 2 hidden layers (3 neurons in each hidden layer)
    - 2 hidden layers (6 neurons in each hidden layer)
  - Try other parameters in MLP to see if better performance can be achieved

Try It!

- Given dataset in “lab2_TryIt.csv”
  - Train **random forest** classifiers on the dataset where the number of trees from 1 to 10
  - Find out the corresponding value of n when the test accuracy is the highest

Try It!

- Given dataset in “lab2_TryIt.csv”
  - Use n SVMs as a base classifier for Bagging on the dataset and calculate test accuracy, n = 2, 4, …,10
  - Try other classifiers (DT, knn)

CLASSIFIER TRAINING/PREDICTION

Training

- After the classifier is defined, the training can start
- `classifier.fit(data, label)`
  - `data` and `label` must be numpy array or list
  - `classifier` will be updated and returned by calling fit()
- `train_set` and `train_label` are numpy arrays in this example

Prediction

Label Prediction

- `classifier.predict(data)`
  - `data`: numpy array or list, contains data of each samples
  - return: 1-D numpy array: labels of each sample

Probability Prediction

- `classifier.predict_proba(data)`
  - `data`: numpy array or list, contains data of each samples
  - return: 2-D numpy array: element[i, j] is the probability of sample i belonging to class j

Training/Prediction

Prediction

```
pred = cls.predict(test_set)
pred
```

```
array([0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1],
      dtype=int64)
```

```
pred_proba = cls.predict_proba(test_set)
pred_proba
```

```
array([[0.58395574, 0.41604426],
       [0.08836755, 0.91163245],
       [0.94146387, 0.05853613],
       ...,
       [0.27435561, 0.72564439],
       [0.23595516, 0.76404484],
       [0.10252132, 0.89747868]])
```

Probability of being class 0

Probability of being class 1

EVALUATION

Evaluation

- Accuracy
- F1-score
- Average precision
- ROC
- AUC
- Confusion matrix

Evaluation

Accuracy

- `from sklearn.metrics import accuracy_score`
- `accuracy_score(true_labels, predicted_labels)`

F1-score

- `from sklearn.metrics import f1_score`
- `f1_score(true_labels, predicted_labels)`

Evaluation

Precision

- `from sklearn.metrics import average_precision_score`
- `average_precision_score(true_labels, predicted_labels)`

Confusion matrix

- `from sklearn.metrics import confusion_matrix`
- `confusion_matrix(true_labels, predicted_labels)`

Evaluation

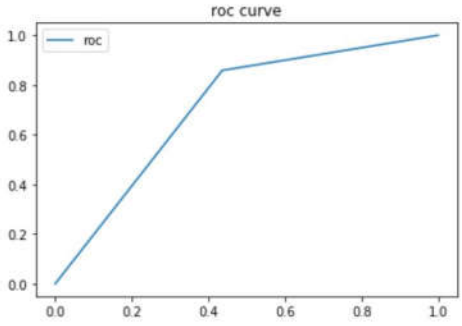
ROC curve

- `from sklearn.metrics import roc_curve`
- `roc_curve(true_labels, predicted_labels)`

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(test_label, pred)
print("fpr:", fpr)
print("tpr:", tpr)
print("thresholds:", thresholds)
```

```
fpr: [0.         0.43636364 1.        ]
tpr: [0.         0.85858586 1.        ]
thresholds: [2 1 0]
```



Evaluation

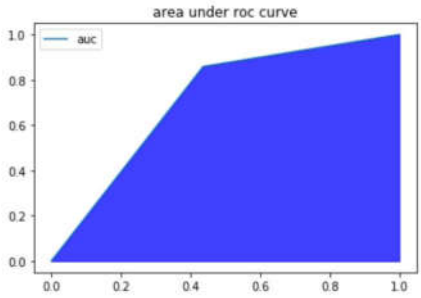
AUC

- `from sklearn.metrics import roc_auc_score`
- `roc_auc_score(true_labels, predicted_labels)`

```
from sklearn.metrics import roc_auc_score

auc = roc_auc_score(test_label, pred)
auc
```

```
0.7111111111111111
```



Evaluation

- Reference:
  https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics

PLOTTING

Plot Decision Boundary

- How to plot a figure?
  1. Construct dense grid that fills the entire space
  2. Predict labels for all points in the grid
  3. Use different colors to represent points with different labels

Example

Bayes classifier decision boundary

1. Construct the grid

```
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

h = 0.01
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

Set the range

Set the interval

```
Z = bayes.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
plt.show()
```


Example

Bayes classifier decision boundary

2. Predict all values in the grid

```
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

h = 0.01
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

Z = bayes.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
plt.show()
```

Example

Bayes classifier decision boundary

3. Colour and plot the grid

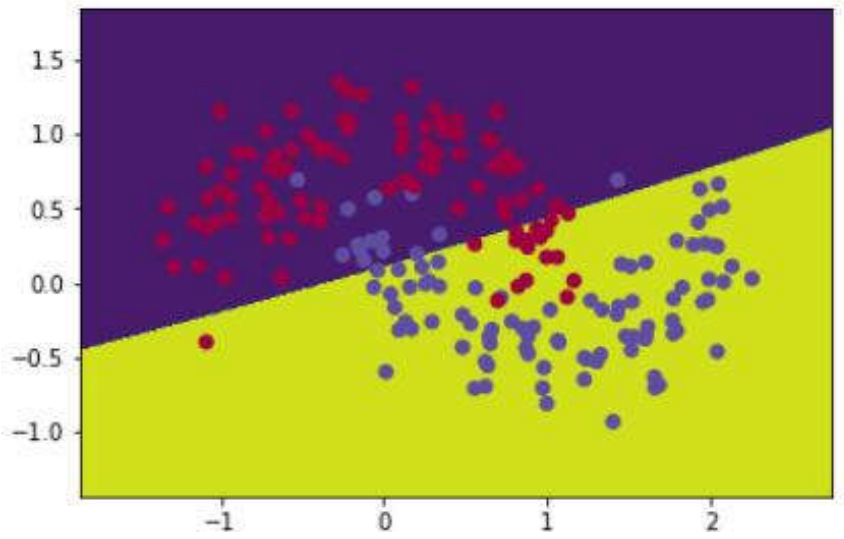
```
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

h = 0.01
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

Z = bayes.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
plt.show()
```


Example



Decision Boundary Plotting

Return values within a range with an interval

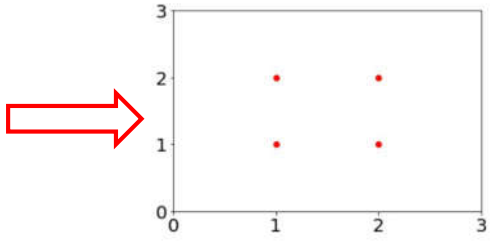
- `np.range([start, ]stop[, step])`
  - `start` and `stop` decide the interval [`start`, `stop`)
  - `step`: spacing between values. If `start` is given, `step` must be given
- e.g. `np.range(1,2,0.25)`
  `= [1,1.25,1.5,1.75]`

Decision Boundary Plotting

Return matrices from coordinate vectors

- `np.meshgrid([`x_1`, `x_2`, .., `x_n`])`
  - x_i: is 1-D array representing the coordinates of a grid, where i=1..n, n is the dimensionality of the coordinate
  - the returned matrices for n coordinate vectors are n-D matrices
- e.g.

```
np.meshgrid(
  array([1,2]),
  array([1,2])
)
```




```
4 points:
(1,2), (2,2)
(1,1), (2,1)
```

Decision Boundary Plotting

Flatten arrays

- `np.ravel(array[, order])`
  `array.ravel([order])`
  - `array` will be flattened into 1-D array
  - `order`: decide how to index, default is to index the elements in row-major
- e.g.
  - `np.ravel(array([ [1,2], [3,4] ]))`

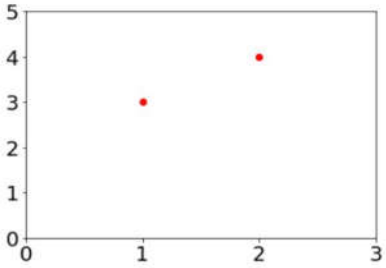
    `array([1,2,3,4])`

Decision Boundary Plotting

Generate a coordinator of a sample

- `np.c_(x1, x2)`
  - `x1` and `x2` are the feature 1 and 2d
- e.g.
  - `np.c_(array([1,2]), array([3,4]))`

    `array([ [1,3], [2,4] ])`



`2 points:`
`(1,3), (2,4)`

Decision Boundary Plotting

Plot contours

- `plt.contourf([x, y, ]z[, colors, cmap])`
  - `x` and `y` are the coordinates of the values in z
  - `z`: the height values over which the contour is drawn
  - `colors`: the colors of the levels, i.e. the lines for contour and the areas for contour
  - `cmap`: a Colormap instance or registered colormap name, maps the level values to colors

Decision Boundary Plotting

Plot points

- `plt.scatter(x, y[, c, cmap])`
  - `x` and `y` are the coordinates of the points
  - `c`: one color for all points or specify colors for each points
  - `cmap`: a Colormap instance or registered colormap name, only used if `c` is an array of floats

UNSUPERVISED LEARNING

Sklearn: Unsupervised learning

- K-means
- PCA

Clustering: K-means

- `from sklearn.cluster import KMeans`
- `kmeans = KMeans([n_clusters, max_iter])`
  - `n_clusters`: int, n clusters, 8 by default
  - `max_iter`: maximum number of iterations, 300 by default

- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Try It!

- Given dataset in “lab2_Try_data.csv”, assume the data is unlabeled
  - Use **k-means** to split the data in to n clusters, where n from 2 to 5
  - Use different colors to plot different clusters, then observe how the value of n effects the results

Clustering: Others

- `SpectralClustering()`
- `DBSCAN()`
- `MeanShift()`

- Reference:
  https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster

PCA

- `from sklearn.decomposition import PCA`
- `pca = PCA([n_components])`
  - `n_components` : int, float, string or None, keep n components

- Reference:
  https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Try It!

- Given dataset in “lab2_TryIt.csv”
  - Apply **k-means** to split the samples in to 2 clusters
  - Use different colors to plot different labeled samples
  - Do this exercise again in the first principle component of PCA