CS M146 - Week 2

Xinzhu Bei

xzbei@cs.ucla.edu

January 20, 2018

Overview

- Miscellaneous
  - Xinzhu Bei, xzbei@cs.ucla.edu
  - Discussion: Friday 2:00 - 3:50 pm, PUB AFF 1337
  - Office Hour: 12pm – 2pm, Mon, Boelter 2432
- Overview
  - LaTeXusage
  - matplotlib.pyplot
  - scikit-learn : installation, documentation
  - K Nearest Neighbors
  - Hint for HW1
  - Lagrangian Multiplier

- Online editors help latexing easier:
  - Overleaf: https://www.overleaf.com/
  - ShareLaTex: https://www.sharelatex.com/
- Latex Editor
  - Lyx: https://www.lyx.org/
  - Texmaker: http://www.xm1math.net/texmaker/
- Latex Symbols

Operators

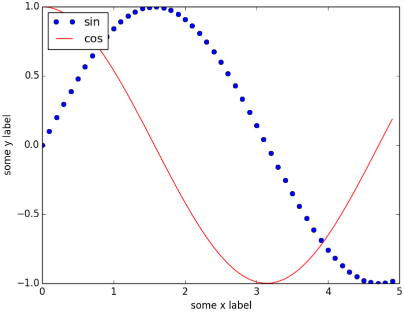
Symbol	Command	Symbol	Command	Symbol	Command
$\pm$	\pm	$\mp$	\mp	$\times$	\times
$\div$	\div	$\cdot$	\cdot	$\ast$	\ast
$\star$	\star	$\dagger$	\dagger	$\ddagger$	\ddagger
$\amalg$	\amalg	$\cap$	\cap	$\cup$	\cup
$\uplus$	\uplus	$\sqcap$	\sqcap	$\sqcup$	\sqcup
$\vee$	\vee	$\wedge$	\wedge	$\oplus$	\oplus
$\ominus$	\ominus	$\otimes$	\otimes	$\circ$	\circ
$\bullet$	\bullet	$\diamond$	\diamond	$\lhd$	\lhd
$\rhd$	\rhd	$\unlhd$	\unlhd	$\unrhd$	\unrhd

- Practice : The probability density of the normal distribution:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

matplotlib.pyplot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 5, 0.1)
y = np.sin(x)
z = np.cos(x)
plt.figure()
plt.plot(x,y,'bo',label='sin')
plt.plot(x,z,'r-', label='cos')
plt.xlabel('some x label')
plt.ylabel('some y label')
plt.legend (loc='upper left')
plt.xlim((0,5))
plt.ylim((-1,1))
plt.show()
```



Numpy, Scipy, scikit-learn

- **Numpy**. Adds Python support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.
- **SciPy** is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.
- **Scikit-learn** is a Python module for machine learning built on top of SciPy and distributed under the 3-Clause BSD license.

- Installation
- Documentation:
  - Decision Tree Classifier:
    http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
  - K-Nearest Neighbor Classifier:
    http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
  - Cross-Validation:
    http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html
  - Metrics:
    http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

- Data
- Baseline Classifier - Majority Vote Classifier

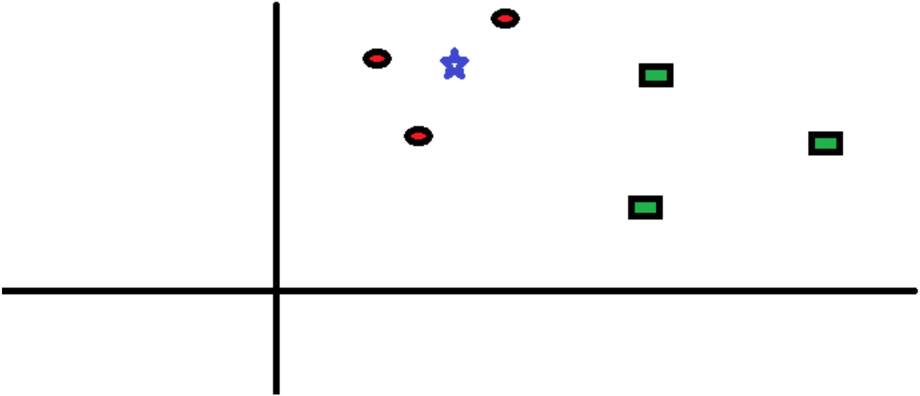
```
class MajorityVoteClassifier(Classifier) :
    def __init__(self) :
        self.prediction_ = None
    def fit(self, X, y) :
        majority_val = Counter(y).most_common(1)[0][0]
        self.prediction_ = majority_val
        return self
    def predict(self, X) :
        if self.prediction_ is None :
            raise Exception("Classifier not initialized. Perform a fit firs
        n,d = X.shape
        y = [self.prediction_] * n
        return y
```

Decision Tree Classifiers

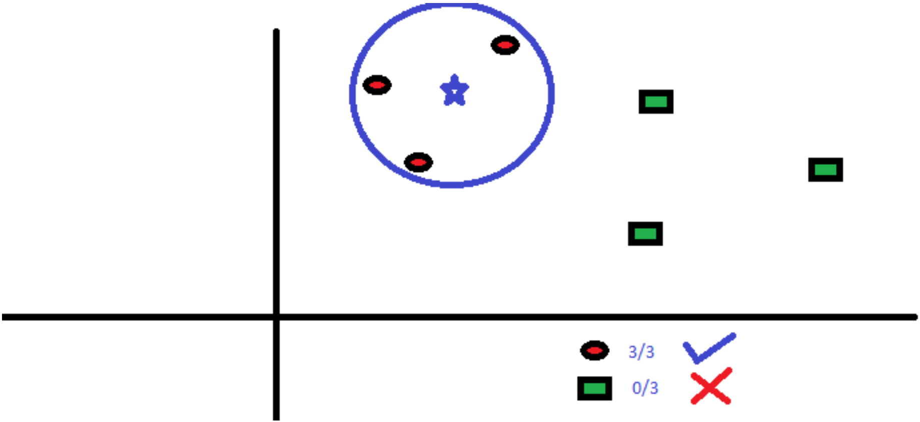
- Selected Attributes:
  - criterion: The function to measure the quality of a split.
  - max_depth: The maximum depth of the tree.
- Seletced Methods:
  - fit(X, y[, sample_weight, check_input,])
    Build a decision tree classifier from the training set (X, y).
  - predict(X[, check_input])
    Predict class or regression value for X.
  - score(X, y[, sample_weight])
    Returns the mean accuracy on the given test data and labels.

```
clf = DecisionTreeClassifier (criterion = "entropy")
clf.fit (X,y)
y_pred = clf.predict(X)
train_error = 1 - clf.score(X, y)
```

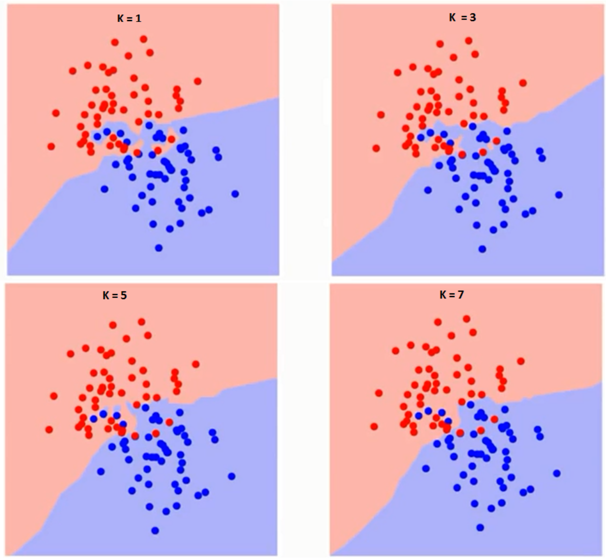
Introduction to K Nearest Neighbors - How does it work?



Introduction to K Nearest Neighbors - How does it work?



How do we choose the factor K?



KNeighborsClassifier

- Selected Attributes:
  - n_neighbors: Number of neighbors to use by default for kneighbors queries.
  - p: Power parameter for the Minkowski metric. When $p = 1$, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for $p = 2$. For arbitrary p, minkowski_distance (l_p) is used.

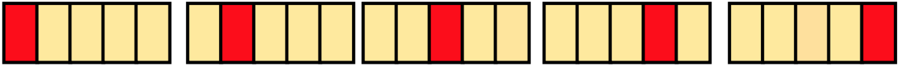
```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
print(neigh.predict([[1.1]]))
print(neigh.predict_proba([[0.9]]))
```

train_test_split

- Selected Attributes:
  - test_size: If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split.
  - random_state: If int, random_state is the seed used by the random number generator.

```python
import numpy as np
from sklearn.cross_validation import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

cross_val_score



- Selected Attributes:
  - estimator : estimator object implementing fit
  - cv : Determines the cross-validation splitting strategy. Integer inputs for cv is to specify the number of folds in a (Stratified)KFold.
  - scoring: (see model evaluation documentation)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)
iris = load_iris()
cross_val_score(clf, iris.data, iris.target, cv=10)
```

sklearn.metrics

The sklearn.metrics module includes score functions, performance metrics and pairwise metrics and distance computations. (See document)

Entropy, Conditional Entropy and Information Gain

- Entropy: If a random variable X has K different values, $x_1, \cdots, x_k$, its entropy is given by

$$H[X] = -\sum_{i=1}^{k} P(X = x_i) \log P(X = x_i)$$

- Conditional Entropy: If $H(Y|X = x)$ is the entropy of the discrete random variable Y conditioned on the discrete random variable X taking a certain value x, then $H(Y|X)$ is the result of averaging $H(Y|X = x)$ over all possible values x that X may take.

$$\begin{aligned} H(Y|X) &\equiv \sum_{x \in \mathcal{X}} p(x)\, H(Y|X = x) \\ &= -\sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x) \end{aligned} \tag{1}$$

Entropy, Conditional Entropy and Information Gain

- The information gain of an attribute a is the expected reduction in entropy caused by partitioning on this attribute

$$\text{Gain}(S, A) = H[S] - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

- In general terms, the expected information gain is the change in information entropy H from a prior state to a state that some information as given:

$$\text{Gain}(S, A) = H[S] - H[S|a]$$

Optimization - Lagrange multipliers

- In mathematical optimization, the method of Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to equality constraints.
- Consider an optimization problem:

$$\begin{aligned} &\text{minimize} && f(x_1, \cdots, x_n) \\ &\text{subject to} && g_k(x_1, \cdots, x_n) = 0, \quad k = 1, \ldots, M \end{aligned} \tag{2}$$

The Lagrangian takes the form

$$\mathcal{L}(x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_M) = f(x_1, \ldots, x_n) - \sum_{k=1}^{M} \lambda_k g_k(x_1, \ldots, x_n)$$

Optimization - Lagrange multipliers

Methods of solving optimizaiton using Lagrangian multipliers:

- Step 1: Solve the following system of equations.

$$\begin{aligned} \frac{\partial L(x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_M)}{\partial x_i} &= 0 \text{ , where } i = 1 \cdots n \\ \frac{\partial L(x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_M)}{\partial \lambda_k} &= 0 \text{ , where } k = 1 \cdots M \end{aligned} \tag{3}$$

$$g_k(x_1, \cdots, x_n) = 0 \text{ , where } k = 1 \cdots M$$

- Step 2:Plug in all solutions $x_1, \cdots, x_n$, from the first step into $f(x_1, \cdots, x_n)$ and identify the minimum and maximum values, provided they exist.

Optimization - Lagrange multipliers

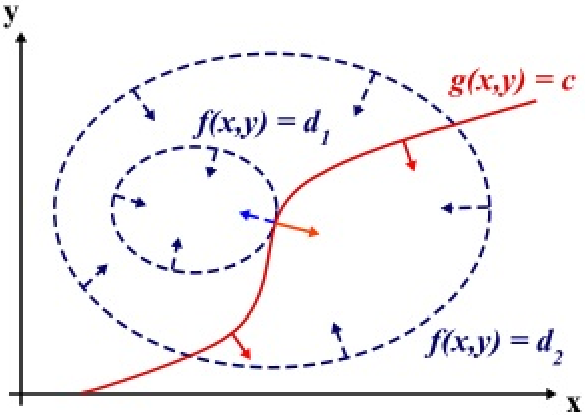
Find the extrema of the function $f(x, y) = 2y + x$ subject to the constraint $0 = g(x, y) = y^2 + xy - 1$.
Solution: Set $\mathcal{L}(x, y, \lambda) = f(x, y) + \lambda g(x, y)$, then

$$\begin{aligned} \frac{\partial L}{\partial x} &= 1 + \lambda y \\ \frac{\partial L}{\partial y} &= 2 + 2\lambda y + \lambda x \\ \frac{\partial L}{\partial \lambda} &= y^2 + xy - 1 \end{aligned} \tag{4}$$

Setting these equal to zero, we see from the third equation that $y \neq 0$, and from the first equation that $\lambda = \frac{-1}{y}$, so that from the second equation $0 = \frac{-x}{y}$ implying that $x = 0$. From the third equation, we obtain $y = \pm 1$.

** Note that it doesn't matter if you are using $f(\cdot) \pm \lambda g(\cdot)$, since all that changes is the sign of λ^*, where $(\lambda^*, x^*, \cdots)$ is the critical point.

Optimization - Lagrange multipliers



The red line shows the constraint $g(x, y) = c$. The blue lines are contours of $f(x, y)$. The point where the red line tangentially touches a blue contour is the maximum of $f(x, y)$, since $d1 > d2$.

The End