Machine Learning

Introduction to Data Analysis in Python
NumPy, Matplotlib, Scikit-learn and Pandas
(seminar 1)

A. Rodomanov

Skolkovo Institute of Science and Technology
3 November 2015

Plan

Array

- Import NumPy:

```
1 import numpy as np
```

- It provides us with a special type for arrays:

```
1 x = np.array([1, -1.5, 2])
```

- Selecting elements:

```
1 print(x[1])
2 print(x[:2])
3 print(x[-1])
```

```
-1.5
[ 1.  -1.5]
2.0
```

- Get the size:

```
1 x.size
```

```
3
```

Array vs List

- It looks like a List but behaves differently:

```
1 x = np.array([1, -1.5, 2])
2 L = [1, -1.5, 2]
3
4 print(2 * x)
5 print(2 * L)
```

```
[ 2. -3.  4.]
[1, -1.5, 2, 1, -1.5, 2]
```

Operations on Arrays

- Array and scalar:

```
1 x = np.array([1, 2, 4])
2 print(x + 1)
3 print(x / 2)
4 print(x**2)
```

```
[2 3 5]
[ 0.5  1.   2. ]
[ 1  4 16]
```

- Array and array:

```
1 x = np.array([1, 2, 4])
2 y = np.array([1, 3, 5])
3 print(x + y)
4 print(x * y)
```

```
[2 5 9]
[ 1  6 20]
```

Operations on Arrays

- Apply a function element-wisely:

```
1 x = np.array([1, 2, 4])
2 print(np.exp(x))
3 print(np.log(x))
```

```
[  2.71828183   7.3890561   54.59815003]
[ 0.          0.69314718  1.38629436]
```

- Some functions return a single number:

```
1 x = np.array([1, -2, 4])
2 print(np.sum(x))
3 print(np.min(x))
```

```
3
-2
```

Multidimensional arrays

- We can create multidimensional arrays (e.g. matrices):

```
A = np.array([[1, 2], [3, 4]])
print(A)
```

```
[[1 2]
 [3 4]]
```

- Using `size`, gives us the number of elements:

```
A.size
```

```
4
```

- To get the dimensions, we must use `shape`:

```
A.shape
```

```
(2, 2)
```

Basic operations

- Slicing:

```
1 A = np.array([[1, 2, 3], [4, 5, 6]])
2 print(A[0, 1])
3 print(A[0])
4 print(A[:, 0])
5 print(A[:, [0, 1]])
```

```
2
[1 2 3]
[1 4]
[[1 2]
 [4 5]]
```

- Transposition:

```
1 print(A)
2 print(A.T)
```

```
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
```

Matrix-matrix operations

- All operators work element-wisely:

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[1, 0], [0, 1]])
print(2 * A)
print(A + B)
print(A * B)
```

```
[[2 4]
 [6 8]]
[[2 2]
 [3 5]]
[[1 0]
 [0 4]]
```

- Note that `A * B` is not the standard matrix-matrix product.
- For matrix-matrix product use `dot`:

```
print(A.dot(B))
```

```
[[1 2]
 [3 4]]
```

Matrix-vector operations

- Matrix-vector product:

```
A = np.array([[1, 2], [3, 4]])
v = np.array([1, -1])
print(A.dot(v))
```

```
[-1 -1]
```

- Note that `A * v` is not the standard matrix-vector product:

```
print(A * v)
```

```
[[ 1 -2]
 [ 3 -4]]
```

- In NumPy matrix + vector = matrix because operations +, *, / etc. between a matrix and a vector involve broadcasting:

```
print(A + v)
```

```
[[2 1]
 [4 3]]
```

Matrix vs Vector

- Matrix with one row/column is not a Vector:

```
A = np.array([[1, 2]])
B = np.array([[1], [2]])
v = np.array([1, 2])
print(A)
print(B)
print(v)
```

```
[[1 2]]
[[1]
 [2]]
[1 2]
```

Matrix vs Vector

- There's no such such thing as row/column vector:

```
1 print(v)
2 print(v.T)
```

```
[1 2]
[1 2]
```

- Compare:

```
1 A = np.array([[1, 2], [3, 4]])
2 v = np.array([1, -1])
3 print(A.dot(v))
4 print(v.dot(A))
```

```
[-1 -1]
[-2 -2]
```

Useful arrays/matrices

- Array of all zeros/ones:

```
1 z = np.zeros(5)
2 print(z)
```

```
[ 0.  0.  0.  0.  0.]
```

- Matrix of all zeros/ones (note double parentheses.):

```
1 A = np.zeros((2, 2))
2 B = np.ones((2, 2))
3 print(A)
4 print(B)
```

```
[[ 0.  0.]
 [ 0.  0.]]
[[ 1.  1.]
 [ 1.  1.]]
```

- Identity matrix: `np.eye(n)`.
- Array $[0, 1, \ldots, n]$: `np.arange(n)`.

Plan

Plotting

- Import Matplotlib:

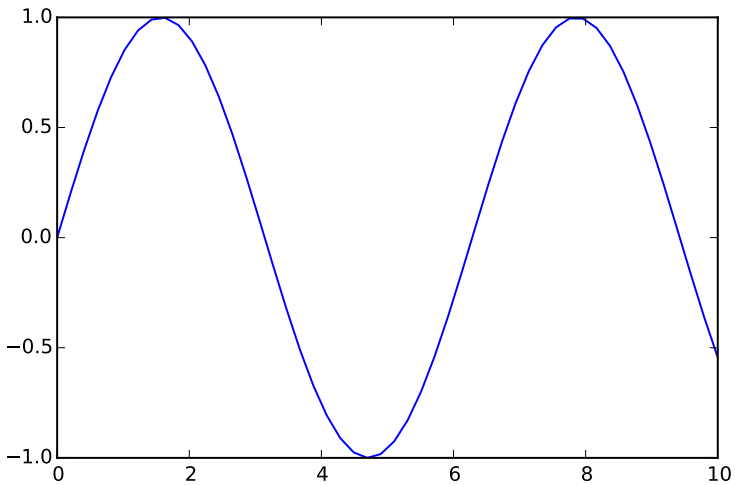
```
1 import matplotlib.pyplot as plt
```

- Automatically insert plots into IPython notebook:

```
1 %matplotlib inline
```

- Basic plot:

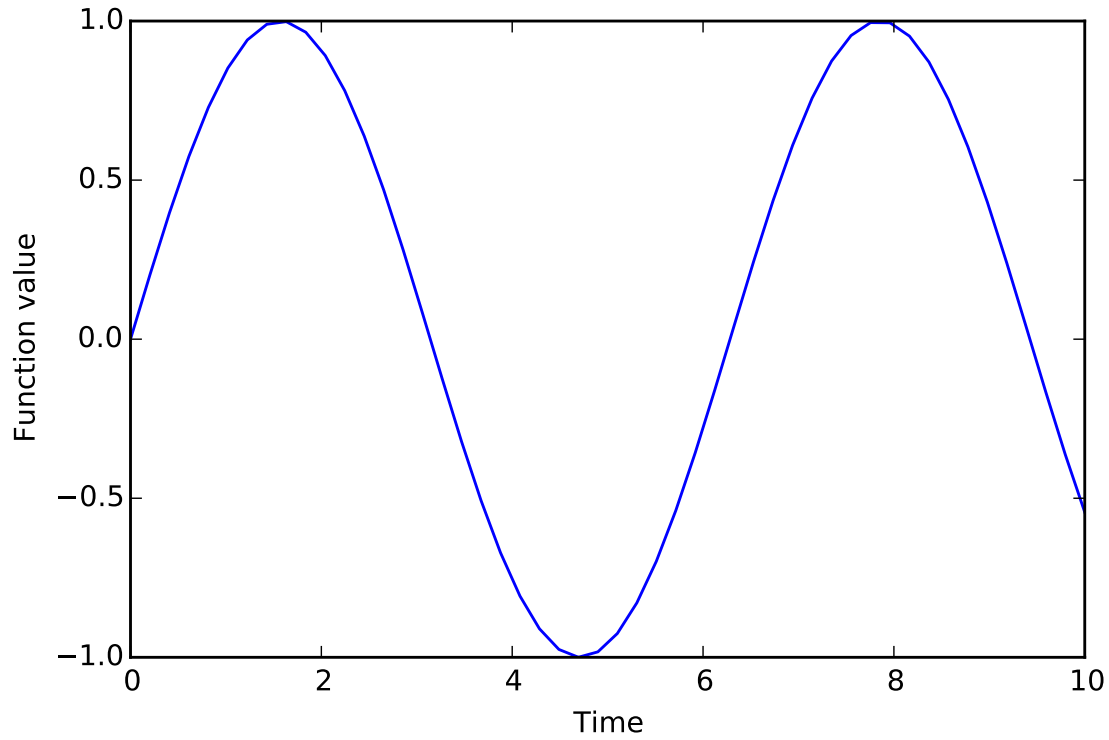
```
1 x = np.linspace(0, 10)
2 y = np.sin(x)
3 plt.plot(x, y)
```



Axis labels

- Always put labels on axes!

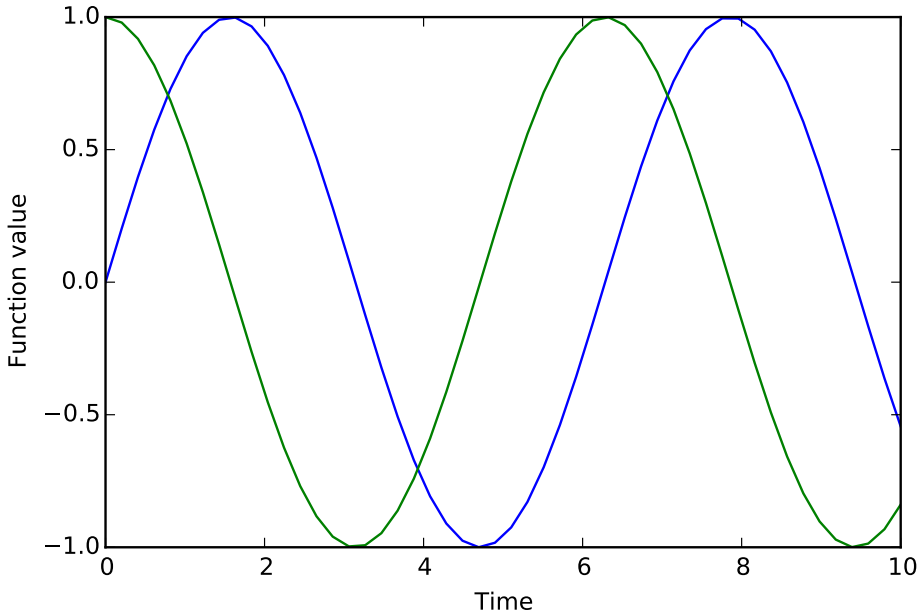
```
1 plt.plot(x, y)
2 plt.xlabel('Time')
3 plt.ylabel('Function value')
```



Several plots

- To draw multiple lines, call `plot` multiple times:

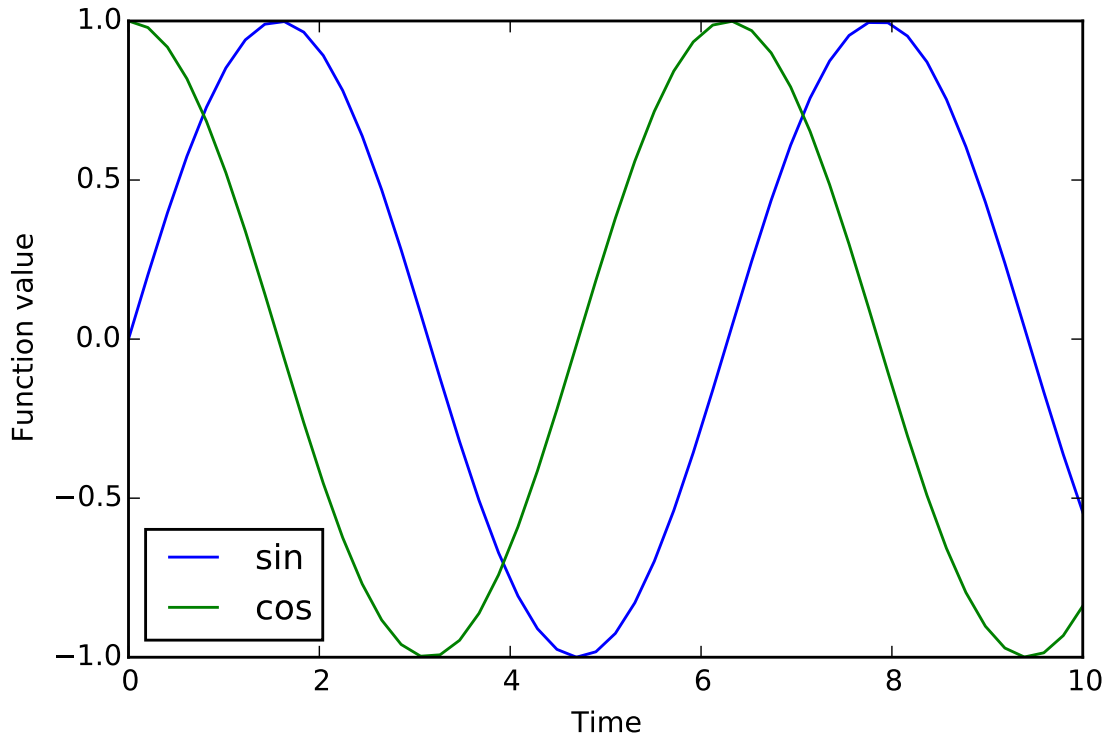
```
1 y = np.sin(x)
2 z = np.cos(x)
3
4 plt.plot(x, y)
5 plt.plot(x, z)
6
7 plt.xlabel('Time')
8 plt.ylabel('Function value')
```



Legend

- Always add a legend when the plot contains multiple lines!

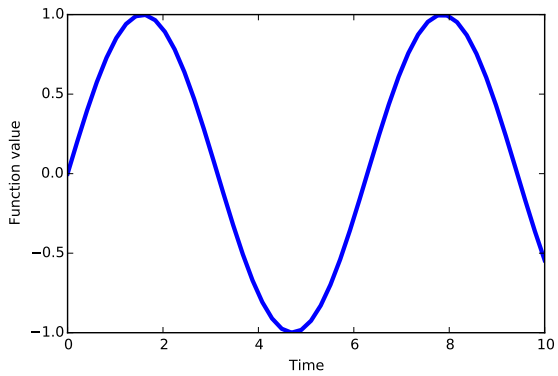
```
1 plt.plot(x, y, label='sin')
2 plt.plot(x, z, label='cos')
3
4 plt.xlabel('Time')
5 plt.ylabel('Function value')
6 plt.legend(loc='best')
```



Line style

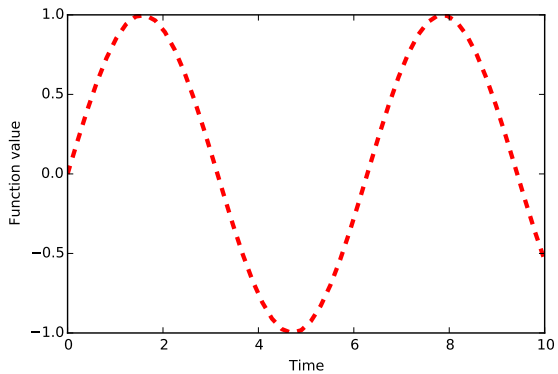
- You almost always want to increase the line width:

```
1 plt.plot(x, y, linewidth=3)
```



- Change colour and line style:

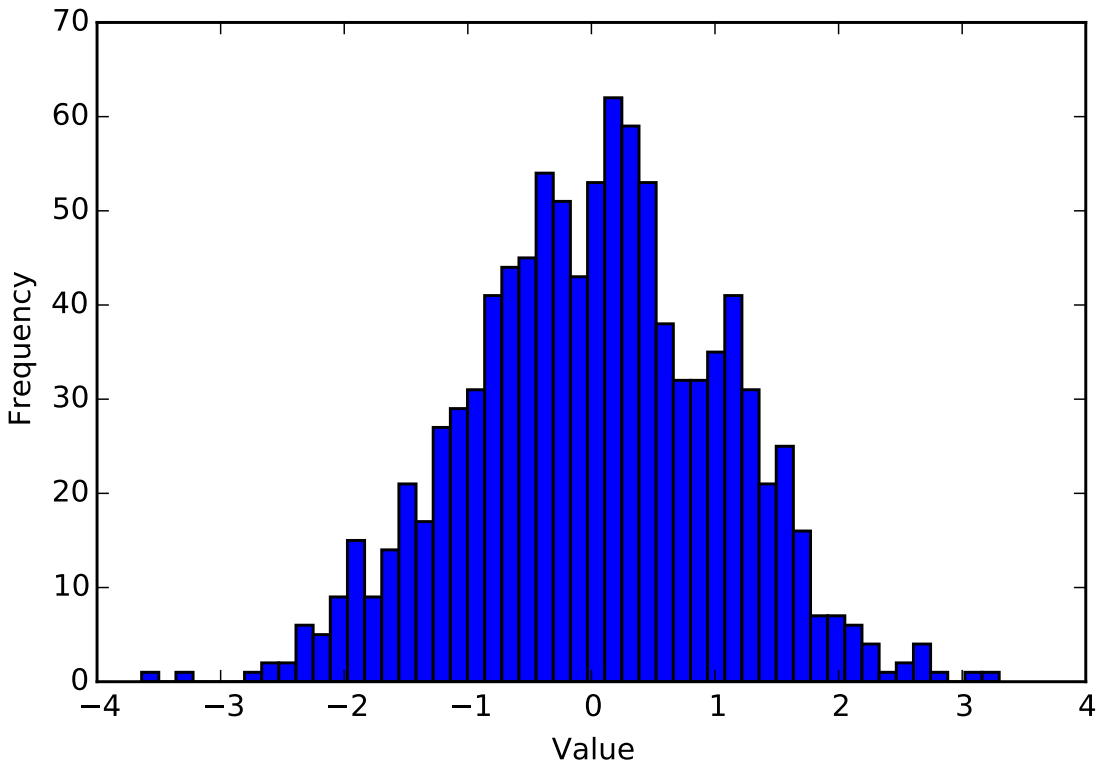
```
1 plt.plot(x, y, linewidth=3, color='r', linestyle='--')
```



Histograms

- For distributions of one-dimensional data you can use histograms:

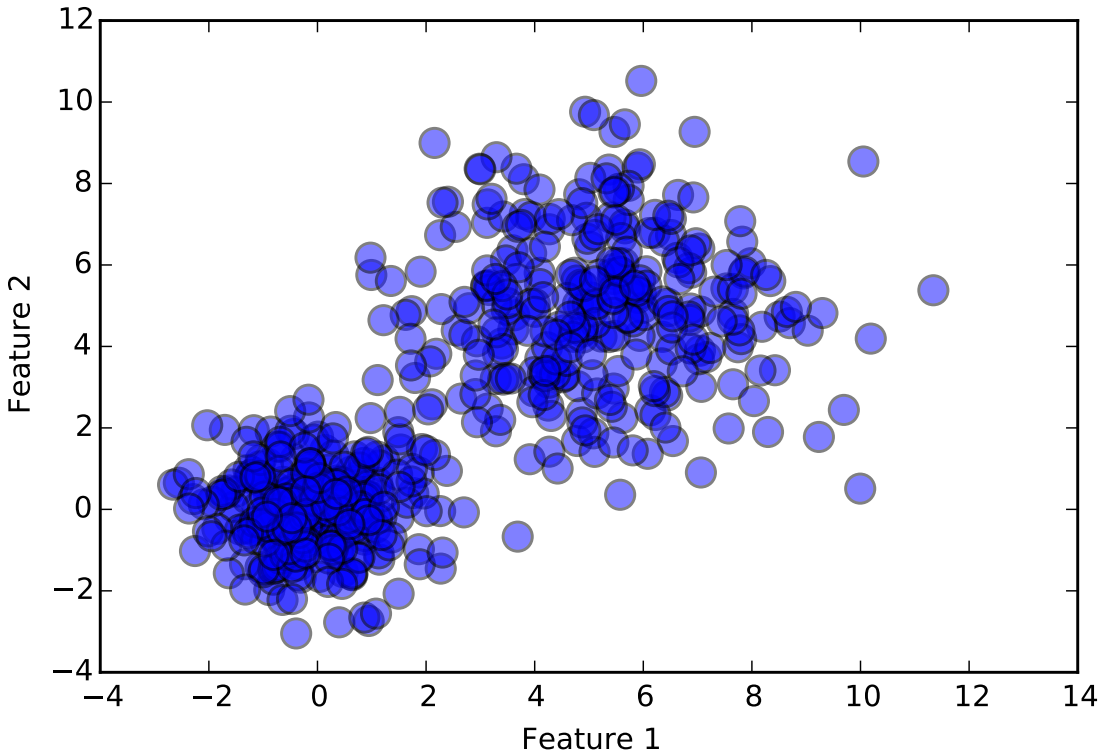
```
1 import numpy.random as npr
2
3 x = npr.randn(1000)
4
5 plt.hist(x, bins=50)
```



Scatter plots

- To visualize two-dimensional data you can use scatter plots:

```
1 X1 = npr.randn(300, 2)
2 X2 = 5 + 2*npr.randn(300, 2)
3
4 X = np.vstack((X1, X2))
5
6 plt.scatter(X[:, 0], X[:, 1], s=100, alpha=0.5)
```

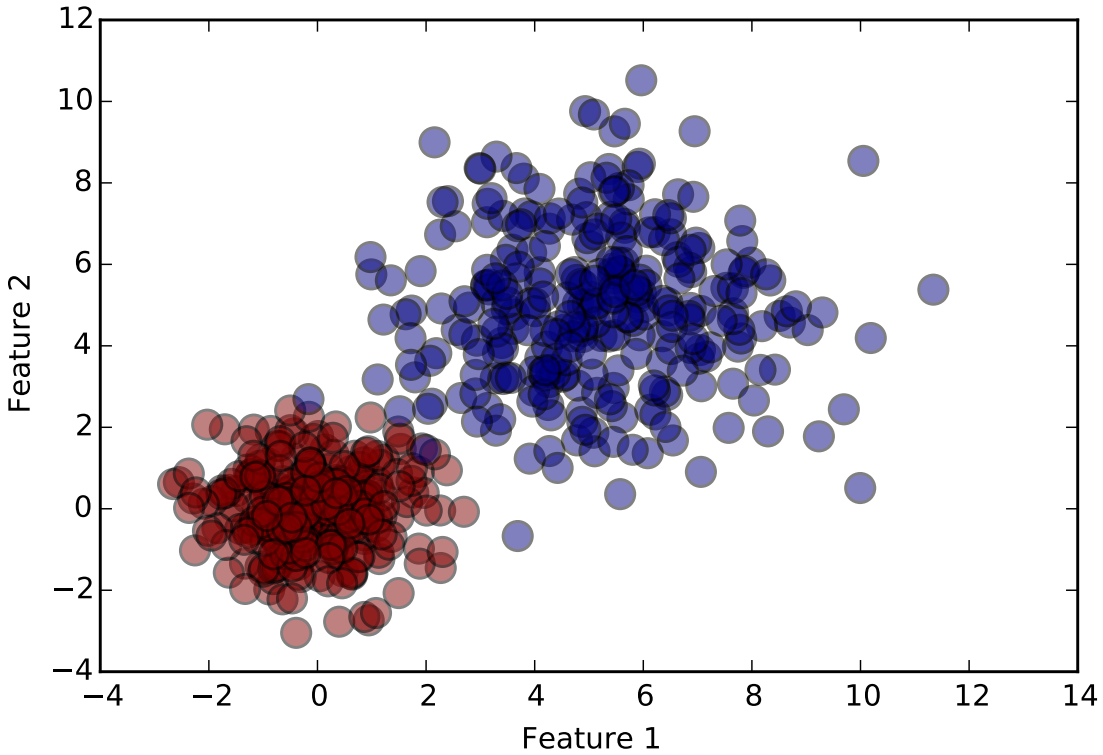


Scatter plots

- Let's paint the two groups in different colours:

```
y1 = np.ones(X1.shape[0])
y2 = -np.ones(X2.shape[0])
y = np.hstack((y1, y2))

plt.scatter(X[:, 0], X[:, 1], c=y, s=100, alpha=0.5)
```



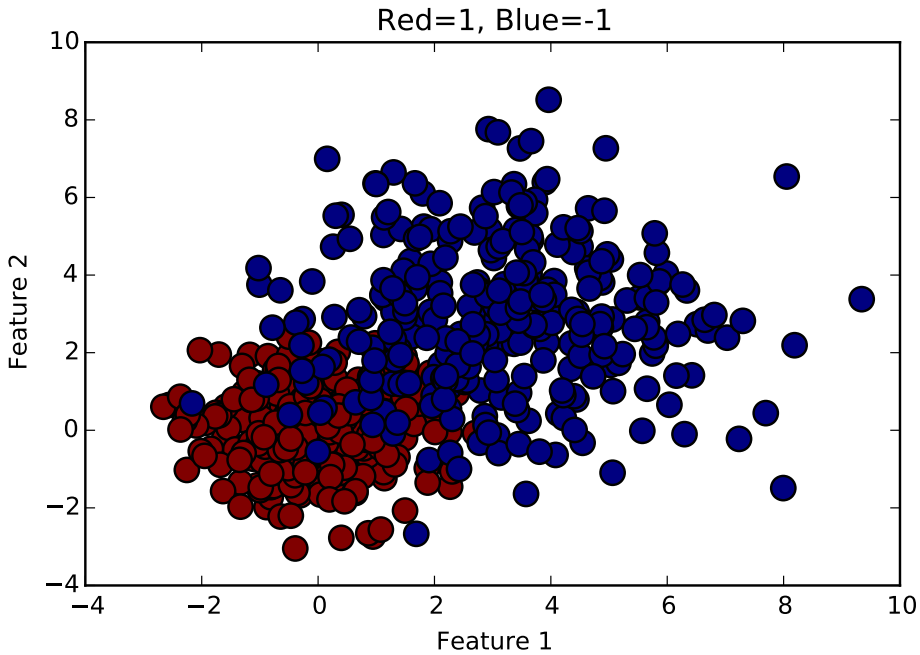
Plan

Generating data

- Toy data:

```
X1 = npr.randn(300, 2)
X2 = 3 + 2*npr.randn(300, 2)
y1 = np.ones(X1.shape[0])
y2 = -np.ones(X2.shape[0])

X = np.vstack((X1, X2))
y = np.hstack((y1, y2))
plt.scatter(X[:, 0], X[:, 1], c=y, s=100)
```



Training a classifier

- Let's train the nearest neighbour classifier:

```
1 from sklearn.neighbors import KNeighborsClassifier as KNN
2
3 clf = KNN(n_neighbors=1)
4 clf.fit(X, y)
```

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

- Let's see what it predicts for the test points $(-4, -4)$ and $(2, 10)$:

```
1 clf.predict([[-4, -4], [2, 10]])
```

```
array([ 1., -1.])
```

Decision boundary

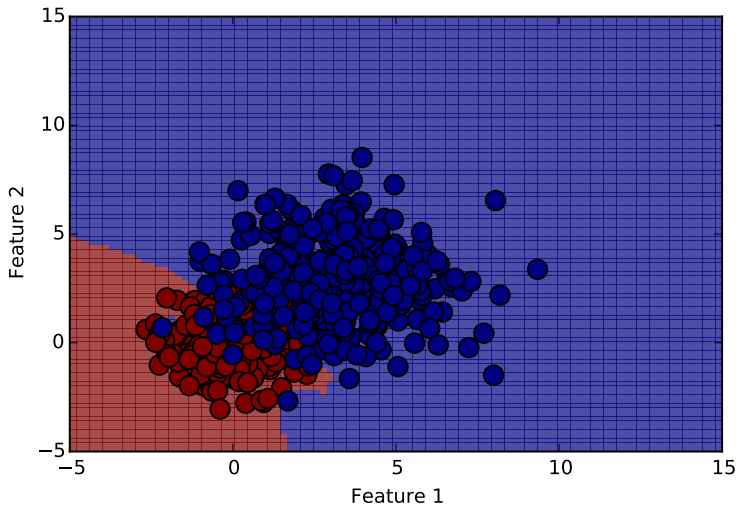
- Auxiliary function for drawing the probability map:

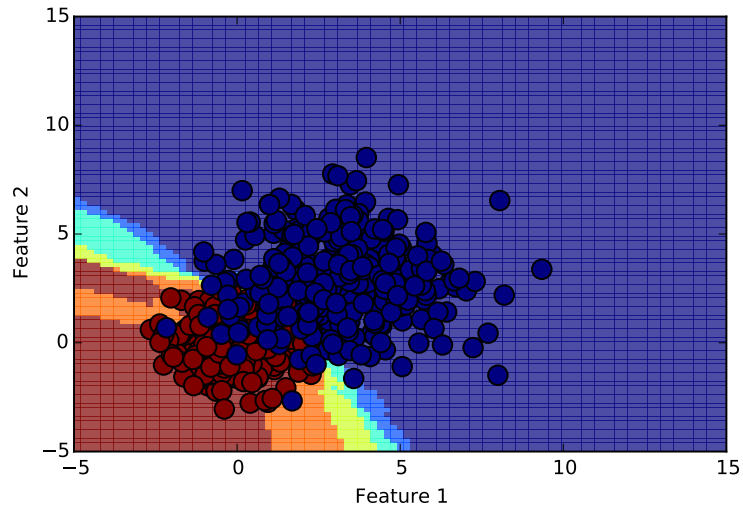
```
1  def proba_map(clf, num=100):
2      f1_min, f1_max = -5, 15
3      f2_min, f2_max = -5, 15
4
5      f1 = np.linspace(f1_min, f1_max, num)
6      f2 = np.linspace(f2_min, f2_max, num)
7      [XX, YY] = np.meshgrid(f1, f2)
8
9      X_proba = np.vstack((XX.ravel(), YY.ravel())).T
10
11     proba = clf.predict_proba(X_proba)[:, 1].reshape(XX.shape)
12
13     plt.pcolormesh(XX, YY, proba, alpha=0.7)
14     plt.scatter(X[:, 0], X[:, 1], c=y, s=100)
```

Varying parameters

- Different number of neighbours:

```
1 proba_map(KNN(n_neighbors=1).fit(X, y))
2 proba_map(KNN(n_neighbors=5).fit(X, y))
```

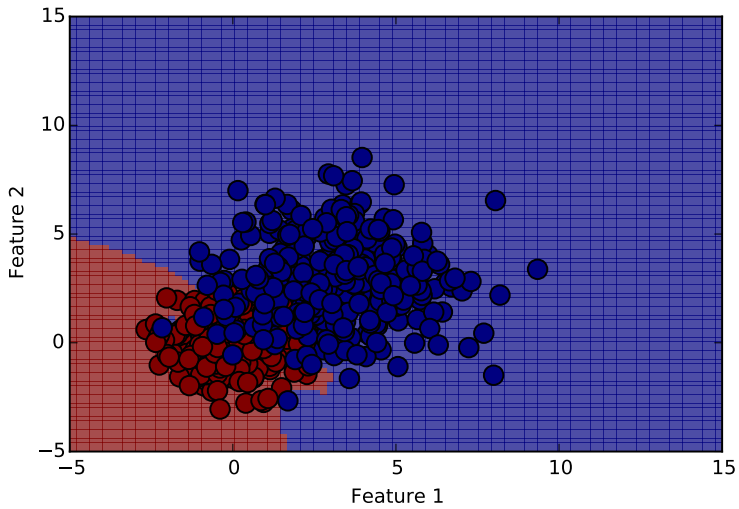


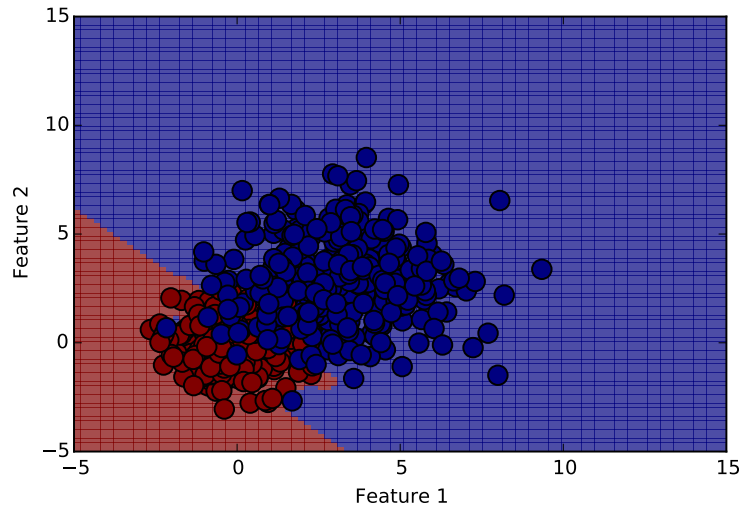


Varying parameters

- Different metric:

```
1 proba_map(KNN(n_neighbors=1).fit(X, y))
2 proba_map(KNN(n_neighbors=1, metric='chebyshev').fit(X, y))
```





Evaluating an algorithm

- Create a cross-validation generator; we'll use stratified 10-fold:

```
from sklearn.cross_validation import StratifiedKFold

cv = StratifiedKFold(y, n_folds=10)
```

- Run cross-validation:

```
from sklearn.cross_validation import cross_val_score

clf = KNN(n_neighbors=1)

scores = cross_val_score(clf, X, y, cv=cv)

print('Minimum score=', np.min(scores))
print('Mean score=', np.mean(scores))
print('Maximum score=', np.max(scores))
```

```
Minimum score= 0.833333333333
Mean score= 0.89
Maximum score= 0.95
```

Choosing best parameters

- To choose the best parameters, we can use grid search:

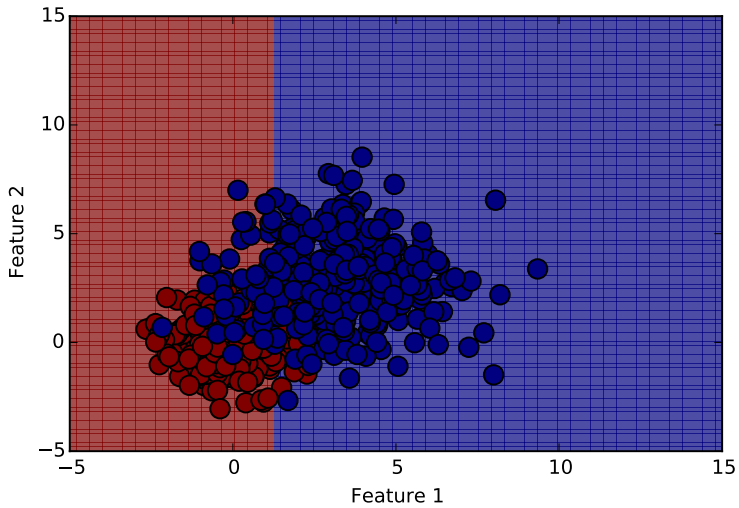
```
1  from sklearn.grid_search import GridSearchCV
2
3  clf = KNN()
4  param_grid = {
5      'n_neighbors': list(range(1, 30)),
6      'metric': ['minkowski', 'chebyshev', 'manhattan']
7  }
8
9  gs = GridSearchCV(clf, param_grid=param_grid, cv=cv)
10 gs.fit(X, y)
11
12 print('Best score: %s' % gs.best_score_)
13 print('Best parameters: %s' % gs.best_params_)
```

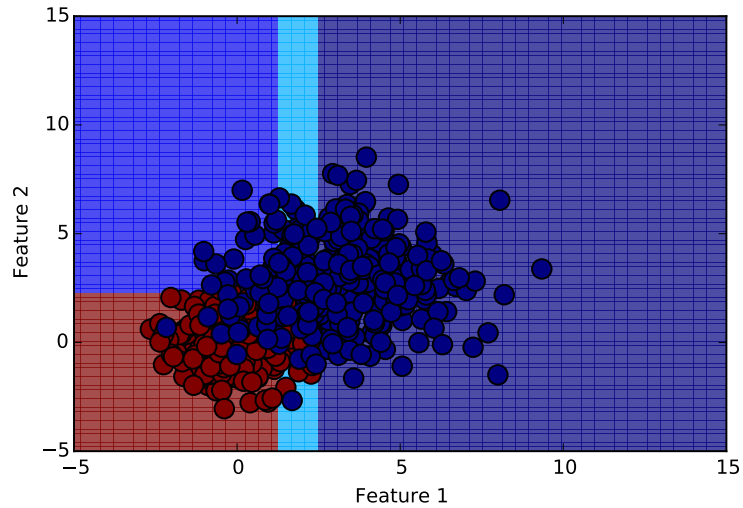
```
Best score: 0.938333333333
Best parameters: {'metric': 'chebyshev', 'n_neighbors': 24}
```

Using a different classifier

- Workflow is the same with a different algorithm:

```
1 from sklearn.tree import DecisionTreeClassifier as DTC
2
3 proba_map(DTC(max_depth=1).fit(X, y))
4 proba_map(DTC(max_depth=2).fit(X, y))
```





Plan

1. NumPy
2. Matplotlib
3. Scikit-learn
4. Pandas

Loading data

- Import Pandas:

```
1 import pandas as pd
```

- Load data and display it:

```
1 df = pd.read_csv('titanic.csv')
2 df.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.0500	NaN	S

- This object is called a DataFrame:

```
1 type(df)
```

```
pandas.core.frame.DataFrame
```

DataFrame

- General information:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 90.5+ KB
```

DataFrame

- Statistics:

```
1 df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Series

- Extract a column:

```
df['Age']
```

```
0      22
1      38
2      26
3      35
...
889     26
890     32
Name: Age, Length: 891, dtype: float64
```

- Each particular column is a Series:

```
type(df['Age'])
```

```
pandas.core.series.Series
```

- We can take its mean, median, min/max etc.:

```
df['Age'].mean()
```

```
29.69911764705882
```

Extracting several columns

- Extract several columns in the same manner:

```
1 df[['Sex', 'Pclass', 'Age']]
```

	Sex	Pclass	Age
0	male	3	22
1	female	1	38
2	female	3	26
3	female	1	35
4	male	3	35
5	male	3	NaN

- This is also a DataFrame:

```
1 type(df[['Sex', 'Pclass', 'Age']])
```

```
pandas.core.frame.DataFrame
```

Filtering

- Get a mask:

```
1 df['Age'] > 60
```

```
0      False
1      False
2      False
3      False
4      False
5      False
...
889     False
890     False
Name: Age, Length: 891, dtype: bool
```

- This is just a Series of booleans:

```
1 type(df['Age'] > 60)
```

```
pandas.core.series.Series
```

Filtering

- Select rows based on condition:

```
1 df[df['Age'] > 60]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
33	34	0	2	Wheadon, Mr. Edward H	male	66.0	0	0	C.A. 24579	10.5000	NaN	S
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5	C
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q
170	171	0	1	Van der hoef, Mr. Wyckoff	male	61.0	0	0	111240	33.5000	B19	S
252	253	0	1	Stead, Mr. William Thomas	male	62.0	0	0	113514	26.5500	C87	S
275	276	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0	1	0	13502	77.9583	D7	S

- This is a DataFrame, so we can do this:

```
1 df[df['Age'] > 60][['Sex', 'Pclass', 'Age', 'Survived']]
```

	Sex	Pclass	Age	Survived
33	male	2	66.0	0
54	male	1	65.0	0
96	male	1	71.0	0
116	male	3	70.5	0
170	male	1	61.0	0

Filtering

- We can use complex conditions using binary operations (&, |, ~):

```
1 df[(df['Age'] > 60) & (df['Sex'] == 'male')]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
33	34	0	2	Wheadon, Mr. Edward H	male	66.0	0	0	C.A. 24579	10.5000	NaN	S
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5	C
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q

Categorical data

- Show all possible values:

```
1 df['Sex'].unique()
```

```
array(['male', 'female'], dtype=object)
```

- Count the number of examples for each value:

```
1 df['Sex'].value_counts()
```

```
male      577
female    314
dtype: int64
```

Missing values

- General info shows us how many values are missing:

```
df.info()
```

```
...
Age              714 non-null float64
SibSp            891 non-null int64
Parch            891 non-null int64
Ticket           891 non-null object
Fare             891 non-null float64
Cabin            204 non-null object
...
```

- Find missing examples in a column:

```
df[df['Age'].isnull()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
26	27	0	3	Emir, Mr. Farred Chehab	male	NaN	0	0	2631	7.2250	NaN	C
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	NaN	Q
29	30	0	3	Todoroff, Mr. Lalio	male	NaN	0	0	349216	7.8958	NaN	S

Missing values

- Remove all examples with missing values:

```
1 df2 = df.dropna()
2 df2.info()
```

```
...
Sex              183 non-null object
Age              183 non-null float64
SibSp            183 non-null int64
Parch            183 non-null int64
Ticket           183 non-null object
Fare             183 non-null float64
Cabin            183 non-null object
Embarked         183 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

- Note that this removes a row if any of its columns contains NaN.

Removing columns

- Remove some columns:

```
df2 = df.drop(['Name', 'Cabin'], axis=1)
df2.head(5)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	male	22	1	0	A/5 21171	7.2500	S
1	2	1	1	female	38	1	0	PC 17599	71.2833	C
2	3	1	3	female	26	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	female	35	1	0	113803	53.1000	S
4	5	0	3	male	35	0	0	373450	8.0500	S

Grouping

- Can join examples into groups and do calculations within groups:

```
1 df.groupby('Sex')['Age'].mean()
```

```
Sex
female    27.915709
male      30.726645
Name: Age, dtype: float64
```

- Can use combinations of groups:

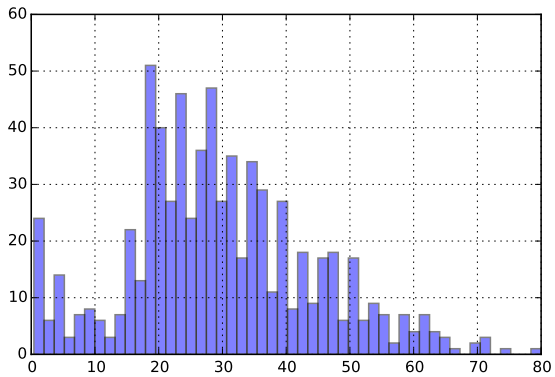
```
1 df.groupby(['Sex', 'Survived'])['Age'].mean()
```

```
Sex     Survived
female  0           25.046875
        1           28.847716
male    0           31.618056
        1           27.276022
Name: Age, dtype: float64
```

Visualization

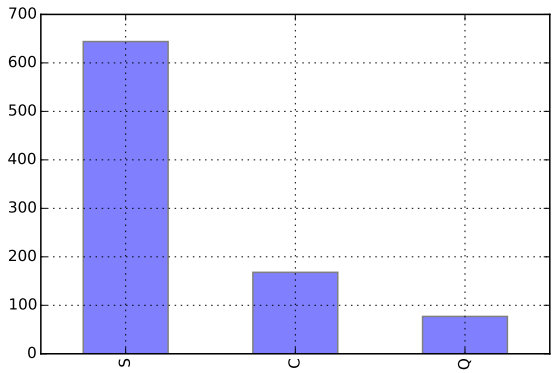
- Histogram:

```
1 df['Age'].hist(bins=50, alpha=0.5)
```



- Bar plot:

```
1 df['Embarked'].value_counts().plot(kind='bar', alpha=0.5)
```



Futher reading and references

- Tutorial on data analysis in Python (Titanic data set):

  https://github.com/savarin/pyconuk-introtutorial.

- Another tutorial on data analysis in Python (Iris data set):

  https://github.com/rhiever/Data-Analysis-and-Machine-Learning-Projects/blob/master/example-data-science-notebook/Example%20Machine%20Learning%20Notebook.ipynb.

- NumPy tutorial:

  https://docs.scipy.org/doc/numpy-dev/user/quickstart.html.

- Matplotlib tutorial:

  http://matplotlib.org/users/pyplot_tutorial.html.

- Sci-kit learn tutorial:

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

- Pandas tutorial:

  http://pandas.pydata.org/pandas-docs/stable/tutorials.html.

- The part about Pandas is based on:

  https://www.kaggle.com/c/titanic/details/getting-started-with-python-ii.