1. Nearest Neighbors:

1.6.1. Unsupervised Nearest Neighbors

`NearestNeighbors` implements unsupervised nearest neighbors learning. It acts as a uniform interface to three different nearest neighbors algorithms: `BallTree`, `KDTree`, and a brute-force algorithm based on routines in `sklearn.metrics.pairwise`. The choice of neighbors search algorithm is controlled through the keyword `'algorithm'`, which must be one of `['auto', 'ball_tree', 'kd_tree', 'brute']`. When the default value `'auto'` is passed, the algorithm attempts to determine the best approach from the training data. For a discussion of the strengths and weaknesses of each option, see Nearest Neighbor Algorithms.

metric : ***string or callable, default 'minkowski'***

metric to use for distance computation. Any metric from scikit-learn or scipy.spatial.distance can be used.

If metric is a callable function, it is called on each pair of instances (rows) and the resulting value recorded. The callable should take two arrays as input and return one value indicating the distance between them. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.

```
from sklearn.neighbors import NearestNeighbors
```

```
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
```

```
nbrs = NearestNeighbors(algorithm='brute').fit(X)
```

```
nbrs
```

```
NearestNeighbors(algorithm='brute', leaf_size=30, metric='minkowski',
         metric_params=None, n_jobs=None, n_neighbors=5, p=2, radius=1.0)
```

```
nbrs.kneighbors(X)
```

```
(array([[0.        , 1.        , 2.23606798, 2.82842712, 3.60555128],
        [0.        , 1.        , 1.41421356, 3.60555128, 4.47213595],
        [0.        , 1.41421356, 2.23606798, 5.        , 5.83095189],
        [0.        , 1.        , 2.23606798, 2.82842712, 3.60555128],
        [0.        , 1.        , 1.41421356, 3.60555128, 4.47213595],
        [0.        , 1.41421356, 2.23606798, 5.        , 5.83095189]]),
 array([[0, 1, 2, 3, 4],
        [1, 0, 2, 3, 4],
        [2, 1, 0, 3, 4],
        [3, 4, 5, 0, 1],
        [4, 3, 5, 0, 1],
        [5, 4, 3, 0, 1]], dtype=int64))
```

2. Join two DataFrame:

```python
df = pd.DataFrame({"key": ["K0", "K1", "K2", "K3", "K4", "K5"],
                   "A": ["A0", "A1", "A2", "A3", "A4", "A5"]})

df
```

	A	key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K3
4	A4	K4
5	A5	K5

```python
other = pd.DataFrame({"key": ["K0", "K1", "K2"],
                      "B": ["B0", "B1", "B2"]})

other
```

	B	key
0	B0	K0
1	B1	K1
2	B2	K2

```python
df.join(other, lsuffix="_caller", rsuffix="_other")
```

	A	key_caller	B	key_other
0	A0	K0	B0	K0
1	A1	K1	B1	K1
2	A2	K2	B2	K2
3	A3	K3	NaN	NaN
4	A4	K4	NaN	NaN
5	A5	K5	NaN	NaN

```python
df.set_index("key").join(other.set_index("key"))
```

key	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2
K3	A3	NaN
K4	A4	NaN
K5	A5	NaN

```python
df.join(other.set_index("key"), on="key")
```

	A	key	B
0	A0	K0	B0
1	A1	K1	B1
2	A2	K2	B2
3	A3	K3	NaN
4	A4	K4	NaN
5	A5	K5	NaN

3. Rename a column:

```python
jack_words.rename(columns = {"count": "count - jack"})
```

	word	count - jack
141	the	24
7	and	15
101	of	12
142	theatre	7
62	for	6

4. Convert a collection of raw documents to a matrix of TF-IDF features:

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

```python
corpus = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]
```

```python
vectorizer = TfidfVectorizer()
```

```python
X = vectorizer.fit_transform(corpus)
```

```python
X
```

```
<4x9 sparse matrix of type '<type 'numpy.float64'>'
	with 21 stored elements in Compressed Sparse Row format>
```

```python
X.shape
```

```
(4, 9)
```

```python
print vectorizer.get_feature_names()
```

```
[u'and', u'document', u'first', u'is', u'one', u'second', u'the', u'third', u'this']
```

```
X.toarray()
```

```
array([[0.        , 0.46979139, 0.58028582, 0.38408524, 0.        ,
        0.        , 0.38408524, 0.        , 0.38408524],
       [0.        , 0.6876236 , 0.        , 0.28108867, 0.        ,
        0.53864762, 0.28108867, 0.        , 0.28108867],
       [0.51184851, 0.        , 0.        , 0.26710379, 0.51184851,
        0.        , 0.26710379, 0.51184851, 0.26710379],
       [0.        , 0.46979139, 0.58028582, 0.38408524, 0.        ,
        0.        , 0.38408524, 0.        , 0.38408524]])
```

5. Generate a random array with 5 rows and 3 columns:

```
np.random.seed(0)

np.random.randn(5, 3)
```

```
array([[ 1.76405235,  0.40015721,  0.97873798],
       [ 2.2408932 ,  1.86755799, -0.97727788],
       [ 0.95008842, -0.15135721, -0.10321885],
       [ 0.4105985 ,  0.14404357,  1.45427351],
       [ 0.76103773,  0.12167502,  0.44386323]])
```

6. Convert True/False to 1/0:

```
doc.dot(random_vectors) >= 0
```

```
array([[False,  True, False, False,  True,  True, False,  True, False,
         True, False, False, False,  True,  True,  True]])
```

```
np.array(doc.dot(random_vectors)>=0, dtype=int)
```

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

7. itertools.combinations(iterable, r): Return r length subsequences of elements from the input iterable.

```
from itertools import combinations
```

```
num_vector = 16
search_radius = 3

for diff in combinations(range(num_vector), search_radius):
    print diff
```

```
(0, 1, 2)
(0, 1, 3)
(0, 1, 4)
(0, 1, 5)
(0, 1, 6)
(0, 1, 7)
(0, 1, 8)
(0, 1, 9)
(0, 1, 10)
(0, 1, 11)
(0, 1, 12)
(0, 1, 13)
(0, 1, 14)
(0, 1, 15)
```