1　k 近傍法 (k-NN)

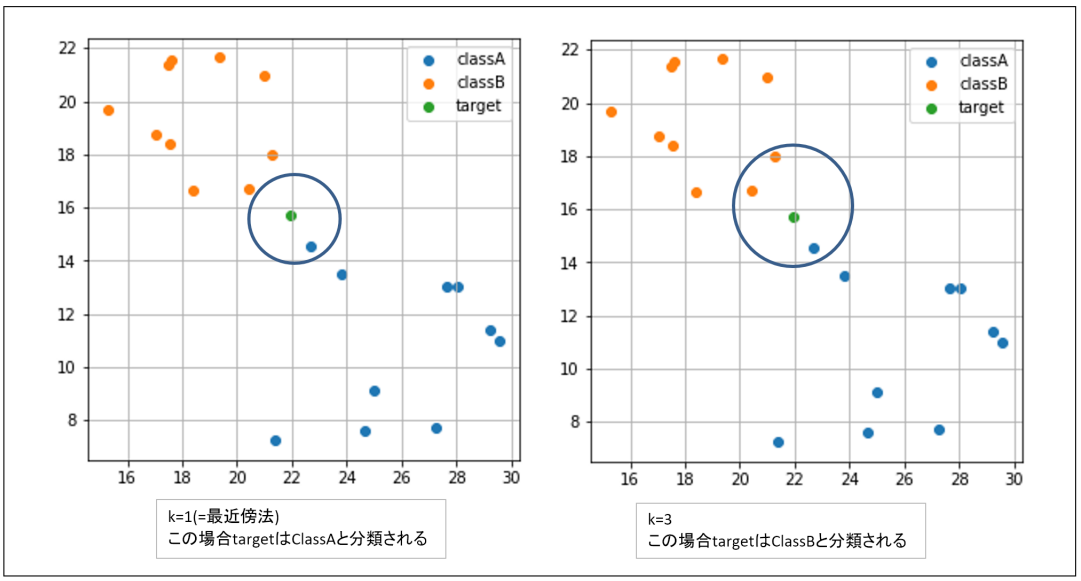
・教師あり分類手法となる

　データ群に対して新しいデータを配置

　→ 近傍より k 個のデータを取得してそれらの所属するクラスをカウント

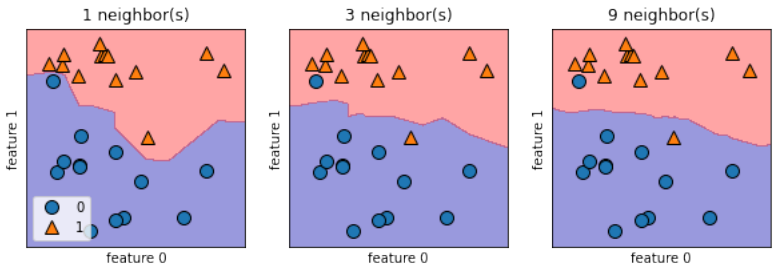
　→ 最頻クラスがその新しいデータのクラスと分類

・k の数=1 の場合は最近傍法といわれる



・k の数が大きいほど決定境界はなめらかになる (k の値が小さいほど過学習気味になる)

　(mglearn.foge データセットで確認 (引用:`https://nanjamonja.net/archives/634#i-3`))



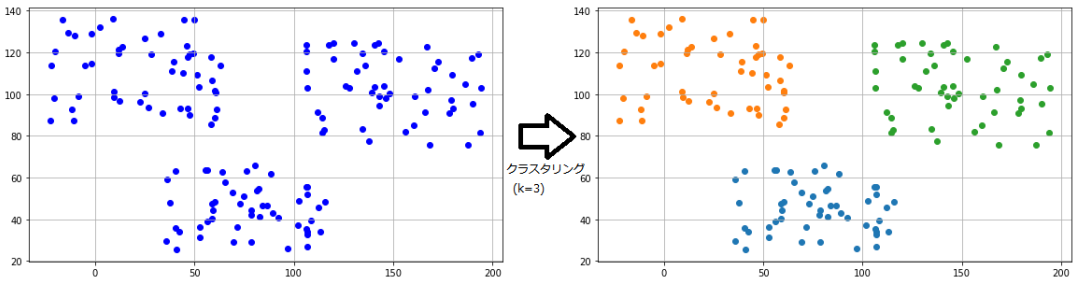
・k の数は最適な値をクロスバリデーション等で決める事になる

2　k 平均法 (k-means)

・教師なし学習、クラスタリング手法となる

　→ クラスタリング=特徴の似ているデータ同士をグループ化する手法

・データ (ラベルなし) を k 個のクラスタに分類する



・k-means のアルゴリズムは以下となる

1) 各クラスタ中心の初期値を設定する

2) 各データ点に対して各クラスタ中心との距離が最も近いクラスタを割り当てる

3) 各クラスタの平均ベクトル (中心) を計算する

4) 収束するまで 2,3 の処理を繰り返す

・k-means 法の欠点

　→ 最初に割り当てるクラスタの位置が不適切だと上手くクラスタリングされない場合がある

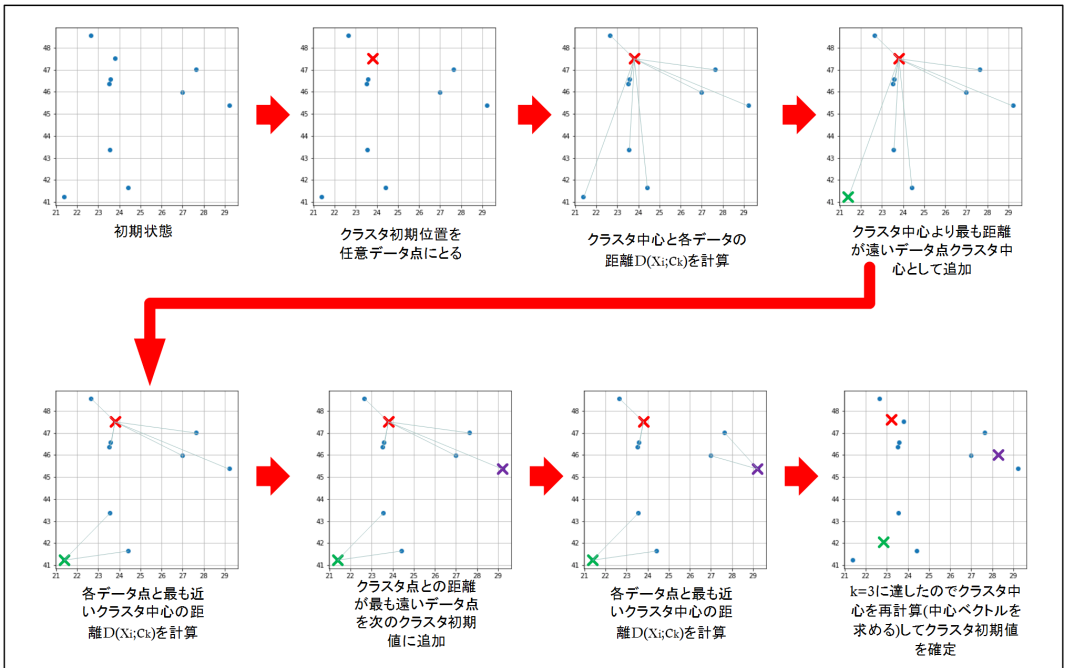
　→k-means 法の欠点はを補う方法として k-means++ 法がある

3　k-means++

・k-means 法のクラスタ初期位置決めをデータ点の重み付き確率分布より決定する方法
　→Wikipedia 等ではそう説明 (※)、他では最遠距離のデータ点を次の初期値に決めると説明
　　→ クラスタ初期位置がお互い近くに配置されないように振り分けなのはたしか
　　→ とは言え、この方法で初期クラスタが完全に離れるわけではない

・クラスタ初期値決定以外の部分は k-means 法と同様のアルゴリズムとなる

・k-means++ 法のクラスタ初期値決定アルゴリズムは以下 (最遠距離選択方法で説明)



k-means++における初期クラスタ位置決定方法(最遠距離、k=3の場合)

(※)
重み付き確率分布 ($\frac{D(x_i;c_k)^2}{\sum_{i=1}^{n} D(x_i;c_k)^2}$) より
ランダム (多分下限を 1 に近い数にリミット?) なデータ位置を次のクラスタ初期値に選択する方法
一説では「外れ値をつかまない様に重み付き確率分布をランダムに選択する」ともあった

4 実習 (k-NN)

ハンズオンコードに対して以下 Cell を追加 (k-NN にて k の値を振り評価)

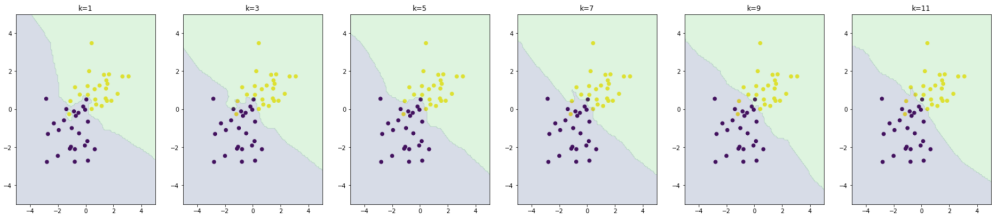
実習(kの値を振ってみる)

In [ ]:

```
def plt_resut2(x_train, y_train, y_pred, ax, n):
    xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
    xx = np.array([xx0, xx1]).reshape(2, -1).T
    ax.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
    ax.contourf(xx0, xx1, y_pred.reshape(100, 100).astype(dtype=np.float), alpha=0.2, levels=np.linspace(0, 1, 3))
    ax.set_title("k=" + str(n))


#奇数を振ってみる
ks = [1,3,5,7,9,11]
# figsize: インチ数
fig, axes = plt.subplots(1,6, figsize=(30, 6))

for n, ax in zip(ks, axes):
  knc = KNeighborsClassifier(n_neighbors=n).fit(X_train, ys_train)
  plt_resut2(X_train, ys_train, knc.predict(xx), ax, n)
```



見た感じではk=9くらいが無難

コードは本ドキュメントと同じレポジトリに提出する

`https://github.com/toruuno/report_ml/blob/master/np_knn.ipynb`

5 実習 (k-means)

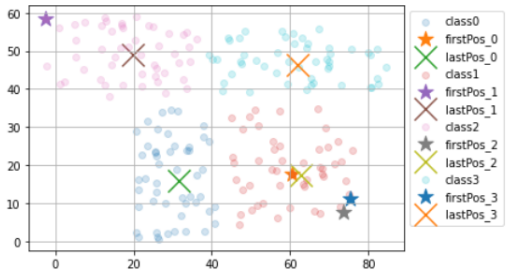
ハンズオンコードに対して k-means++ の実装を追加 (以下は成功例の Cell)

```
#k-means++法(収束するまで)
centers_, initialCList=kMeans_pp(i_max=100, X_t=X_data, seed_=4)
print(initialCList)
print(centers_)

#確定したクラスタを見てみる(成功例、だいたい成功する)
#初期クラスタと確定クラスタの比較
plt.grid()
for c in range(csize):
  plt.scatter(data[c][0],data[c][1],label="class"+str(c),alpha=0.2)
  plt.scatter(X_data[initialCList[c]][0],X_data[initialCList[c]][1],marker='*',label='firstPos_' + str(c), s=200)
  plt.scatter(centers_[c][0],centers_[c][1],marker='x',label='lastPos_' + str(c), s=400)

plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
```

```
[60, 107, 84, 89]
[[31.58937657 15.95611794]
 [19.80769019 48.92428909]
 [62.67447846 17.5486146 ]
 [61.88543785 46.23426712]]
<matplotlib.legend.Legend at 0x7fde5974fe50>
```



コードは本ドキュメントと同じレポジトリに提出する

`https://github.com/toruuno/report_ml/blob/master/np_kmeans.ipynb`