

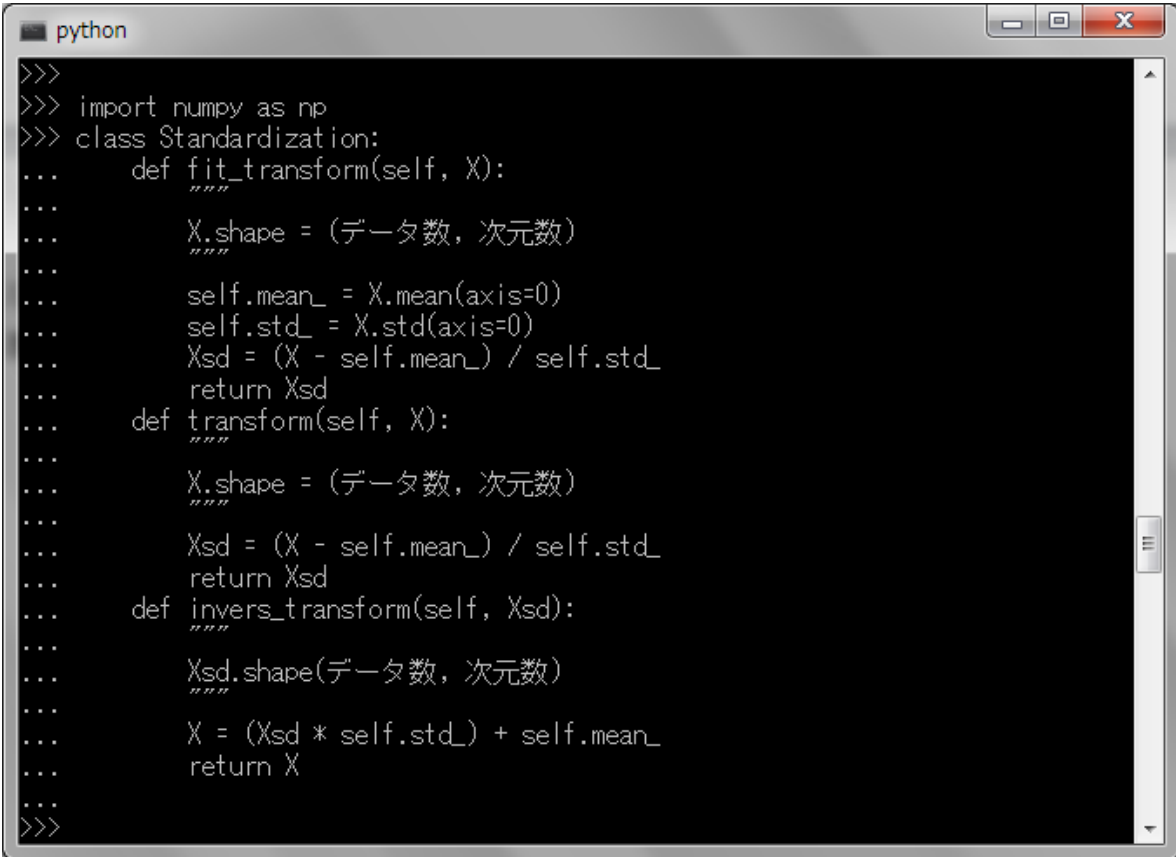
目 次

内容

1.	線形回帰モデル.....	2
1. 1	データの標準化の実装.....	2
2.	非線形回帰モデル.....	3
3.	ロジスティック回帰モデル.....	4
3. 1	シグモイド関数と負の対数尤度関数	4
4.	主成分分析.....	5
5.	アルゴリズム	6
5. 1	k-means.....	6
6.	サポートベクターマシン.....	6

1. 線形回帰モデル

1. 1 データの標準化の実装



```
>>>
>>> import numpy as np
>>> class Standardization:
...     def fit_transform(self, X):
...         """
...         X.shape = (データ数, 次元数)
...         """
...         self.mean_ = X.mean(axis=0)
...         self.std_ = X.std(axis=0)
...         Xsd = (X - self.mean_) / self.std_
...         return Xsd
...     def transform(self, X):
...         """
...         X.shape = (データ数, 次元数)
...         """
...         Xsd = (X - self.mean_) / self.std_
...         return Xsd
...     def invers_transform(self, Xsd):
...         """
...         Xsd.shape(データ数, 次元数)
...         """
...         X = (Xsd * self.std_) + self.mean_
...         return X
>>>
```

2. 非線形回帰モデル

XORの実装

```
python
>>> def AND(x1, x2):
...     w1,w2,theta=0.5,0.5,0.7
...     tmp=x1*w1+x2*w2
...     if tmp <= theta:
...         return 0
...     elif tmp > theta:
...         return 1
...
>>> def NAND(x1, x2):
...     x = np.array([x1,x2])
...     w = np.array([-0.5,-0.5])
...     b = 0.7
...     tmp = np.sum(w*x) + b
...     if tmp <= 0:
...         return 0
...     else:
...         return 1
...
>>> def OR(x1, x2):
...     x = np.array([x1,x2])
...     w = np.array([0.5,0.5])
...     b = -0.2
...     tmp = np.sum(w*x) + b
...     if tmp <= 0:
...         return 0
...
...     else:
...         return 1
...
>>> def XOR(x1,x2):
...     s1 = NAND(x1,x2)
...     s2 = OR(x1,x2)
...     y = AND(s1,s2)
...     return y
...
>>> XOR(0,0)
0
>>> XOR(1,0)
1
>>> XOR(0,1)
1
>>> XOR(1,1)
0
>>> 
```

3. ロジスティック回帰モデル

3. 1 シグモイド関数と負の対数尤度関数

```
python
>>> import numpy as np
>>> class Sigmoid:
...     def forward(self, x, w, b):
...         """
...         x.shape(データ数, 次元数)
...         w.shape(1,次元数)
...         """
...
...         self.x= x
...         z= np.sum(w*x, axis=1)+b
...         Y_pred= 1/(1+np.exp(-z))
...         self.Y_pred= Y_pred
...         return Y_pred
...     def backward(self, dy):
...         """
...         dy.shape = (データ数, )
...         """
...
...         dz=dy*(1.0-self.Y_pred)*self.Y_pred
...         dw=np.sum(dz.T * self.x, axis=0)
...         db= np.sum(dz)
...         return dw, db
>>> class NegativeLogLikelihood:
...     def forward(self, Y_pred, Y_true):
```

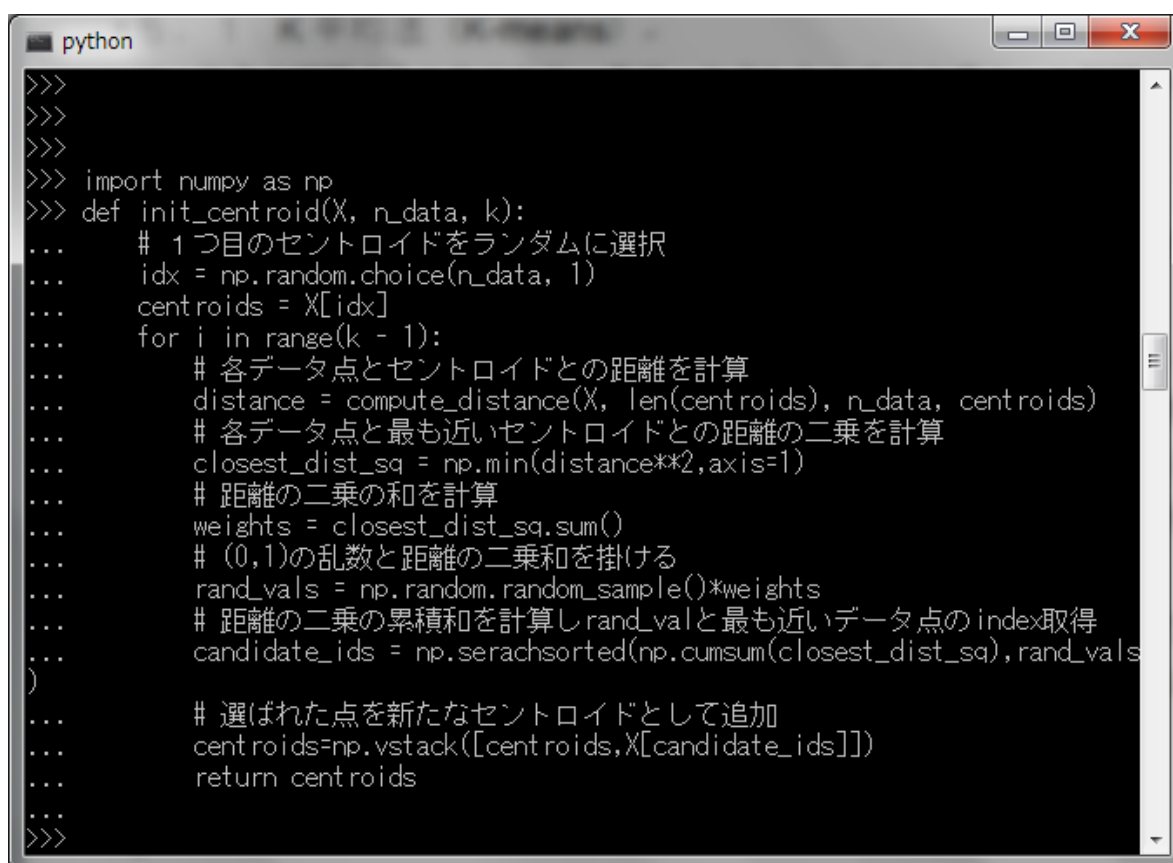
```
python
...     """
...     Y_pred.shape(データ数,)
...     Y_true.shape(データ数,)
...     """
...
...     self.Y_pred= Y_pred
...     self.Y_true= Y_true
...     loss= - (Y_true*np.log(Y_pred)+(1-Y_true)*np.log(1-Y_pred))
...     return loss.sum()
...     def backward(self):
...         dy= - (self.Y_true/self.Y_pred)+((1-self.Y_true)/(1-self.Y_pred))
...         return dy
...
>>> ■
```

4. 主成分分析

```
python
>>> import numpy as np
>>> def pca(X, n_components):
...     # データから平均を引く
...     X = X - X.mean(axis=0)
...
...     # 共分散行列の作成
...     cov = np.cov(X, rowvar=False)
...
...     # 固有値と固有ベクトルを計算
...     l, v = np.linalg.eig(cov)
...
...     # 固有値の大きい順に固有ベクトルを並べる
...     l_index = np.argsort(l)[::-1]
...     v_ = v[:, l_index]
...
...     # n_components分、主成分方向を取得する
...     components = v_[:, :n_components]
...
...     # データを低次元空間へ射影
...     T = np.dot(X, components)
...
...     return T
>>>
```

5. アルゴリズム

5. 1 k-means



```
python
>>>
>>>
>>> import numpy as np
>>> def init_centroid(X, n_data, k):
...     # 1つ目のセントロイドをランダムに選択
...     idx = np.random.choice(n_data, 1)
...     centroids = X[idx]
...     for i in range(k - 1):
...         # 各データ点とセントロイドとの距離を計算
...         distance = compute_distance(X, len(centroids), n_data, centroids)
...         # 各データ点と最も近いセントロイドとの距離の二乗を計算
...         closest_dist_sq = np.min(distance**2,axis=1)
...         # 距離の二乗の和を計算
...         weights = closest_dist_sq.sum()
...         # (0,1)の乱数と距離の二乗和を掛ける
...         rand_vals = np.random.random_sample()*weights
...         # 距離の二乗の累積和を計算しrand_valと最も近いデータ点のindex取得
...         candidate_ids = np.argsort(np.cumsum(closest_dist_sq),rand_vals
... )
...         # 選ばれた点を新たなセントロイドとして追加
...         centroids=np.vstack([centroids,X[candidate_ids]])
...         return centroids
...
>>>
```

6. サポートベクターマシン