ML HW02

Author	陳懷安
Student ID	0753736

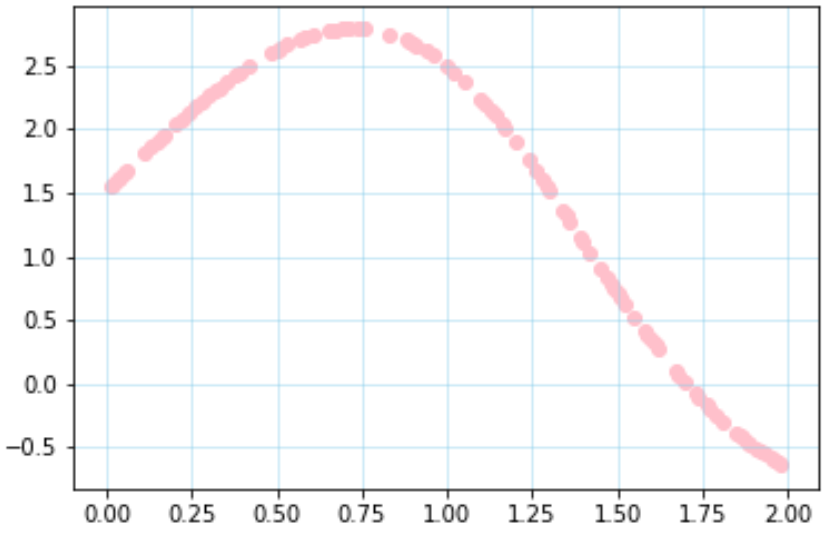
Sequential Bayesian Learning

Preview

題目所提供的資料集中含有 **100 個 `x` 以及 100 個 `t`** 。

並要求透過貝氏估計法更新估計權重，逼近真實分配參數。

下面我先將給定之資料集以散布圖的形式繪製出來，發現呈現一凹向原點(Concave)的趨勢。



Given Dataset

Data Processing

題目給定Design Matrix的形式：

$$\phi_j(x) = \sigma(\frac{x-\mu_j}{s}),\ j = 0, 1, 2, ..., (M-1)$$

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

$$\Phi = [\phi_0\ \phi_1\ ...\ \phi_{M-1}]^T$$

Bayesian Learning

以下將逐次分別放入`[5, 5, 20, 50]`筆數據，進行模型的訓練。

1-(1)

我們有Predictive distribution：

$$p(t|x,\mathbf{x},\mathbf{t}) = N(t|y(x,\mathbf{w}),\beta^{-1})$$

其中，

$$m(x) = \beta\phi(x)^T\mathbf{S}\sum_{n=1}^{N}\phi(x_n)t_n$$
$$s^2(x) = \beta^{-1} + \phi(x)^T\mathbf{S}\phi(x)$$
$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\phi(x_n)\phi(x)^T$$

依次放入訓練資料後，算得的`t`的一個變異數的區間分布。

這邊我們設定`alpha = 10E-6`, `beta = 1`繪製出的圖形如下。

1-(2)

接著我們利用 Conjugate：

$$p(w) = N(w|0,10^0\mathbf{I})$$

Posterior Distribution：

$$p(w|\mathbf{t}) = \mathbf{N}(\mathbf{w}|\mathbf{m_N},\mathbf{S_N})$$

其中，

$$S_N^{-1} = S_0^{-1} + \beta\mathbf{\Phi}^T\mathbf{\Phi}$$
$$m_N = S_N\beta\mathbf{\Phi}^T\mathbf{t}$$

第二小題當中，我們設定`alpha = 1`, `beta = 1`，計算出`posterior distribution`。

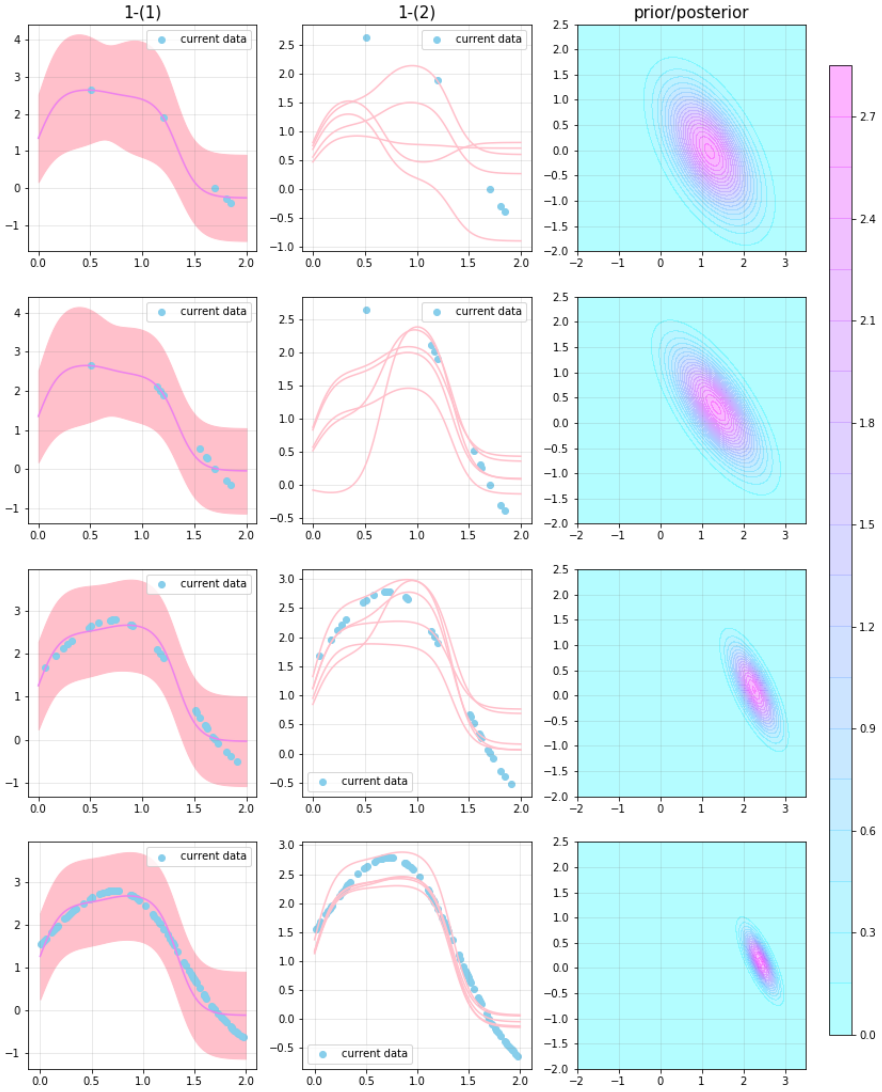
接著從該分配中隨機抽了`5`組樣本，並繪製出估計的分配，如粉紅色線所示。

1-(3)

這邊我選擇了`w0,` `w1`兩個系數的分配來繪製等高線圖。

我們都知道，多元常態分配的子集合所成的分配也會是多元常態分配。因此直接調用三元常態當中的平均及共變異矩陣即可得到`w0, w1`之二元常態分配參數。

以此我將二元常態的pdf直接作用在座標點上，以等高線圖的方式呈現。



Classification — Logistic Regression

Preview

從題目得知，這次拿到的是`5`個人各`10`張照片。同樣先來檢視一下資料型態。

```
0 b'P5\n'
1 b'92 112\n'
2 b'255\n'
3 b'sqqvsusurtvrrststssturp}|~\x82\x82\x86\x7f\x8f\x7f1fd\\W`XUU_aESTNIBKB?G9B?B=<
OcqkSUjonoponmdtolnrmmloqkqmronnrurwsnustuuquuqsrqvssuoqlupsqtnYga\\bf^c`\\MHSMDLMA
5:=>=8:7>08<TqhQUdslqokgholmropqloormmmqlpmttussttpwsusttpttqstqshhca\\fbOT]dfhb]X
LMT@=B;EG>7/,/7:)-3006<l`QW`pllqfjlsnkppntkmoppkppjqpvqsttvqtotrvquoxqtsrruokaXUFIX
aeqmfTOKI9:3>=9)7.?...\' 5*+-(.1.^UPMV]Z]afkpnqmoqnpjonoommqnnourrrvuotqpvpusousrus
qseQIDLIQZYb\\RSDI@782/2*;%)$5\')+"\x171"-)),(/CNFHTTQMTY_omokmqnnopqnrmnmmnlurtpuq
uvqvouvuutqvtnsXE<JVhwrnhm[SE>@8/,+"4)"\x1c\'\x1e#\'\x1c#\x1f\x15!\x1f/ (%%(;FETPQS
LIQJ[nlpikqomrmmnoqlsjqwttvurrtqurqwqrstrpYM=?AZhtx`aZVIF>A?=7)%$!!\x1f\x16\x1e\x1f
\x1d"\x1e\x1d\x1f\x15\x1e\x1e*) ##$4GCMTTWVXZKHYonpppqlqjponlqqnoutvsutqstuovsxtswj
```

可看到圖檔內容如上

照片為`.pgm` `p5`格式，尺寸為`92*112`的為灰度圖。

前五張照片如下所示：

Data Processing

將每個人的照片隨機挑出`5`張作為訓練資料，剩餘`5`張作為測試資料，並將十進位數據存入矩陣中。

Modeling — `Gradient Descent`

本次將實作`Logistic Regression`，並用以分類人臉歸屬。由於本次為一多類別的分類問題，顧會先將**類別標籤**透過`one-hot encoding`的方式處理，並且會在`feature`中加入`bias`項。

權重更新的方式則會採用`gradient descent`進行。

$$p(C_1|\phi) = \sigma(\mathbf{w}^T\phi)$$
$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n$$

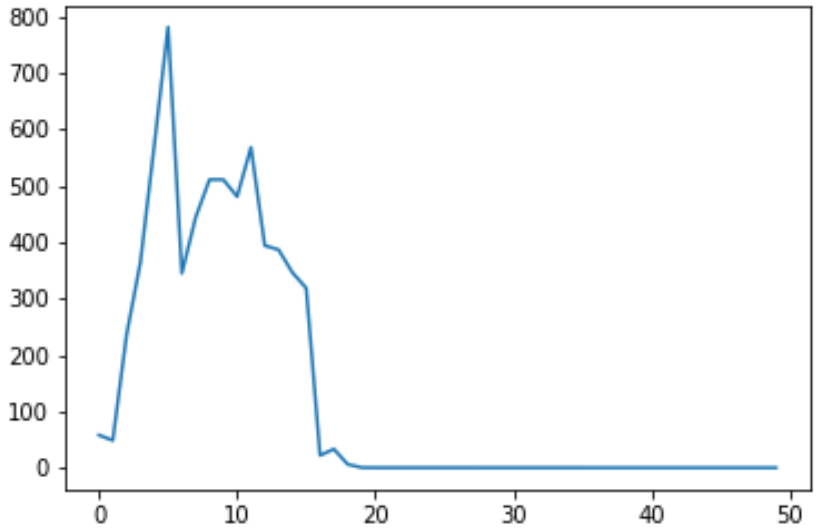
更新如下：

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - \nabla E(\mathbf{w})$$

Training

實際進入訓練階段，這邊我以`0.001`作為權重更新的學習率；並迭代`50`次。

每次迭代都會計算當次的`cross entropy`，並繪製線圖如下。



Predict

並且實際放入測試資料集後，預測出的結果如下：

```
[35]: print(np.argmax(y_p, axis=1))
      [0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4]
```

完全預測正確。

Modeling — `Iterative Reweighted Least Squares`

這階段的建模，將加入`PCA`作為`features`的縮減，接著權重更新的方式採用`Newton-Raphson algorithm`。

更新如下：

$$\begin{aligned}\mathbf{w}^{(new)} &= \mathbf{w}^{(old)} - \mathrm{H}^{-1}\nabla E(\mathbf{w}) \\ &= \mathbf{w}^{(old)} - (\Phi^T\mathbf{R}\Phi)^{-1}\Phi^T(\mathbf{y}-\mathbf{t})\end{aligned}$$

$$\mathbf{R}_{nn} = y_n(1-y_n)$$

PCA

首先我先計算出共變異矩陣(Covariance Matrix)，

$$Cov(\mathrm{X}) = \mathrm{E}\left[(\mathbf{X} - \mathrm{E}[\mathbf{X}])\,(\mathbf{X} - \mathrm{E}[\mathbf{X}])^{\mathrm{T}}\right]$$

接著計算出特徵值及特徵向量，再依照特徵值大小作排序，以此為引，取出前幾特徵向量，乘上原本的資料達到構面縮減的效果。

Training

將縮減過的資料(`2`, `5`, `10`個主成分)分別放入模型中，迭代次數設定為`200`次，並同樣繪製出`cross entropy`的走勢圖。

Predict

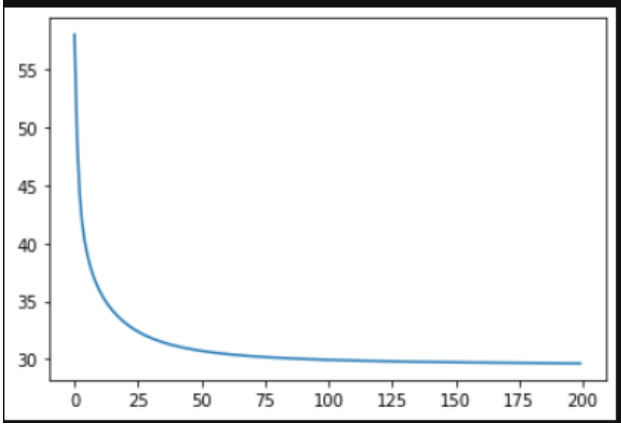
- `2`個主成分

```
y2 = predict(w_IRLS_2, pca_test_2, 'SGD')
print(np.argmax(y2, 1))
```

```
[0 1 0 0 0 1 1 1 1 1 2 2 2 2 2 2 0 2 1 1 0 1 1 0 0]
```

```
plt.plot(CE_IRLS_2)
```

```
[<matplotlib.lines.Line2D at 0x1f11a760748>]
```



圖中可看出，`loss` 優化曲線平滑，但預測結果多不正確。

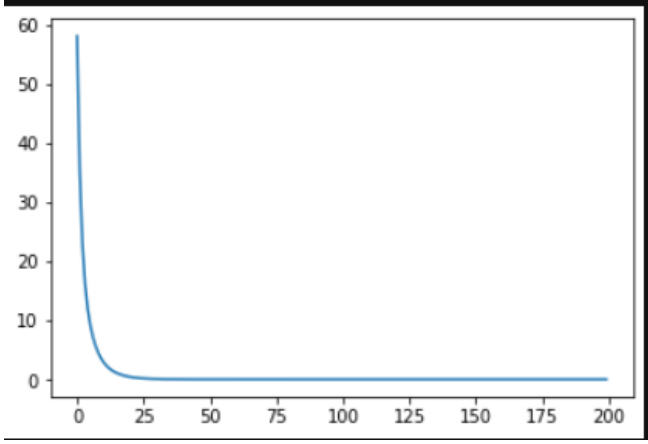
- `5` 個主成分

```
y5 = predict(w_IRLS_5, pca_test_5, 'SGD')
print(np.argmax(y5, 1))
```

```
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 4 4 4 4 4 4 4 4 4 4]
```

```
plt.plot(CE_IRLS_5)
```

```
[<matplotlib.lines.Line2D at 0x1f11ad11088>]
```



約莫在25次左右收斂，但最終無法分辨`第四類`及`第五類`的差別。

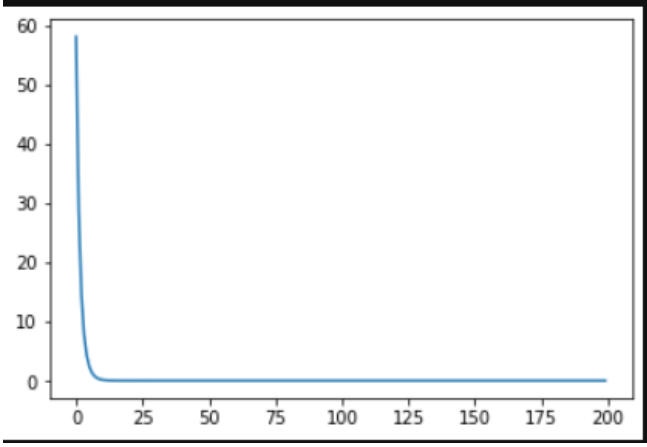
- `10` 個主成分

```
y10 = predict(w_IRLS_10, pca_test_10, 'SGD')
print(np.argmax(y10, 1))
```

```
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4]
```

```
plt.plot(CE_IRLS_10)
```

```
[<matplotlib.lines.Line2D at 0x1f11e06b8c8>]
```



約在前10次就已收斂至最終樣貌，且預估結果也非常精準。

Discussion

以上兩種演算法，可以發現到在迭代過程中，`cross entropy`的走勢在採用`Newton-Raphson`明顯平滑許多，雖然收斂速度在這無從比較(放入的`features`數量不同) 。可以看出在`w`的更新過程，這個方法是更細緻的，也比較不會有「**走過頭**」的情形。

而單在第二段的演算法當中，置入不同數量的主成分—`2, 5, 10` ，也顯示了不同的收斂結果。其中`2`個主成分的建模過程，收斂速度慢，最後的`loss`也維持在相對較高點的位置，預測出的結果有許多錯誤歸類；`10`個主成分的那個模型，不僅收斂速度快，且`loss`也最終趨近於0(分類完全正確)；`5`個主成分的模型在各方面均居中。

Bonus

k-Nearest Neighbors

透過`KNN`算法將pokemon **type**做分類，原始資料如下：

	Name	Type 1	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	Porygon-Z	Normal	0.922711	0.317304	0.290731	0.171038	1.806401	0.176067	0.628458	0.610715	False
1	MeowsticFemale	Psychic	0.349722	0.007835	-0.734330	0.407399	0.307119	0.391714	1.094794	1.863870	False
2	Aipom	Normal	-0.530522	-0.526704	-0.029600	-0.419865	-0.932672	-0.542758	0.461909	-0.642440	False
3	Froakie	Water	-0.912515	-0.920574	-0.478064	-1.010768	-0.298360	-0.938112	-0.004427	1.863870	False
4	Slaking	Normal	2.043777	2.145988	2.853384	1.352844	0.653107	-0.183346	0.961555	-0.015863	False
...	...	...	...	...	...	...	...	...	...	...	...
153	Oshawott	Water	-0.962340	-0.526704	-0.510098	-0.813801	-0.269528	-0.902171	-0.870481	1.237292	False
154	Horsea	Water	-1.070295	-1.230044	-0.990595	0.171038	-0.067701	-1.620996	-0.370834	-1.269018	False
155	Alomomola	Water	0.382939	2.567992	0.130566	0.564973	-0.932672	-0.902171	-0.204286	1.237292	False
156	Prinplup	Water	-0.156834	-0.273501	-0.157733	0.092251	0.249455	0.212008	-0.703932	0.610715	False
157	Solosis	Psychic	-1.111815	-0.808040	-1.310926	-1.010768	0.941431	-0.722464	-1.703224	1.237292	False

158 rows × 11 columns

Data Processing

先將資料區分為`training data`(前`120`筆)、`testing data`(後`38`筆)，接著一除非數值欄位，以利距離計算。

Prediction Steps

1. 計算testing data 與前120筆資料的`Euclidean Distance`。

	row120	row121	row122	row123	row124	row125	row126	row127	row128	row129	...	row148	row149	row150
0	4.312744	2.161868	3.035997	3.234835	4.578133	2.650847	4.271452	1.522128	3.533382	5.137002	...	2.060960	3.644503	3.571996
1	5.254549	3.017154	3.212303	3.876190	4.037371	2.535061	4.211904	2.421893	3.630429	4.880200	...	1.871566	3.177615	3.314400
2	4.575334	2.812710	3.738525	2.335577	1.948120	1.617383	1.104631	3.667497	0.915888	1.841176	...	2.563300	2.170445	4.730220
3	6.000604	4.188437	2.641573	3.857958	2.271896	3.082800	3.361334	3.690683	3.498327	3.383777	...	2.462205	1.795941	5.535217
4	4.010242	4.027451	6.194219	5.326217	6.685885	4.035400	5.907997	3.251117	5.112435	6.979991	...	4.079654	6.104002	4.475686
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
115	4.843778	3.857398	6.250130	5.614307	7.357980	4.599867	6.590061	3.444396	5.732949	7.732460	...	4.456084	6.485184	2.773717
116	3.114872	1.678406	3.259609	1.856593	3.934921	2.458583	3.153005	2.218773	2.908601	4.323369	...	2.839425	3.114559	3.912478
117	5.596912	4.610546	3.212034	3.369393	1.380797	3.803562	2.682973	4.898113	3.341744	2.375329	...	4.105795	1.977954	6.860564
118	5.562381	4.109124	4.367746	2.740437	2.730329	3.110808	2.039143	4.644890	2.629436	2.902416	...	4.215120	3.006834	5.495166
119	5.700456	4.035953	2.703631	3.449968	1.466755	2.867166	2.753655	3.884017	3.018401	2.722631	...	2.705233	1.327016	5.638121

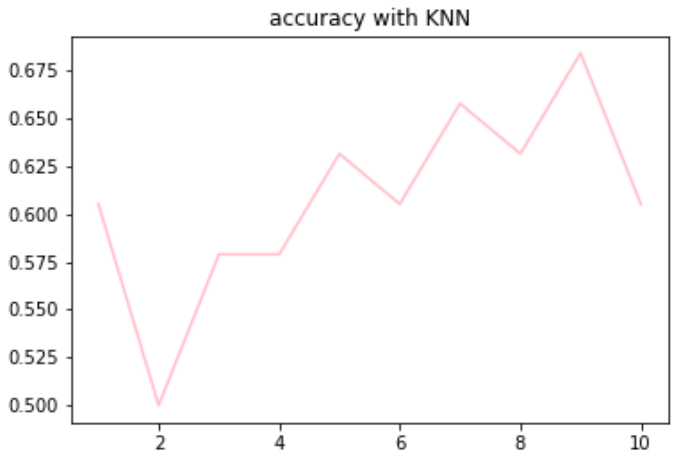
120 rows × 38 columns

2. 依序將各欄做距離遠近的排序，並取前`k`近的資料作為參照。
3. 將前`k`筆資料中出現次數最多的`type`作為預測結果。
   這邊的作法是將`list`元素作為`dictionary`的`key`，元素出現的次數作為`value`。

💡 如果遇到元素出現次數相當的情形，將參照 `R` 語言 `caret` 套件內 `KNN` 演算法的應對方式，**隨機抽取**平票元素的其中一種。

結果如下：

index	values
0	0.6052631579
1	0.5
2	0.5789473684
3	0.5789473684
4	0.6315789474
5	0.6052631579
6	0.6578947368
7	0.6315789474
8	0.6842105263
9	0.6052631579



PCA + KNN

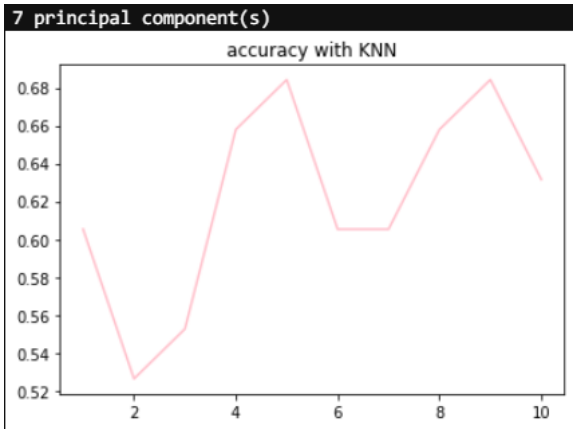
Data Processing

`PCA` 的部分沿用前面所用的方式，將數值變數所成矩陣套入後，求取主成分。

接著同樣計算後 `38` 筆資料與前 `120` 筆資料的距離。

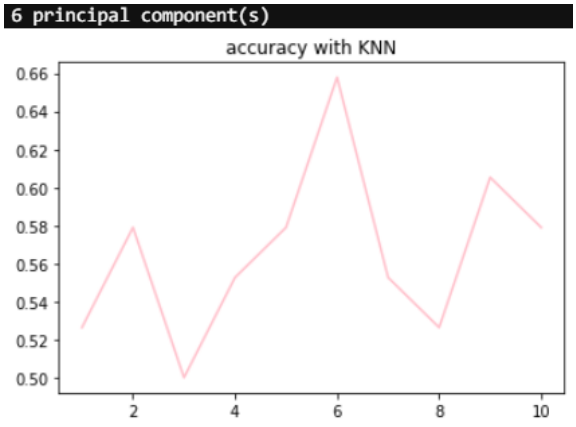
Result

- **Dimension 7**




```
[0.6052631578947368,
 0.5263157894736842,
 0.5526315789473685,
 0.6578947368421053,
 0.6842105263157895,
 0.6052631578947368,
 0.6052631578947368,
 0.6578947368421053,
 0.6842105263157895,
 0.631578947368421]
```

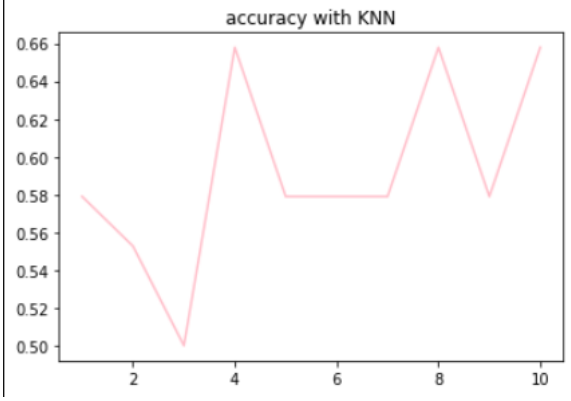
- **Dimension 6**




```
[0.5263157894736842,
 0.5789473684210527,
 0.5,
 0.5526315789473685,
 0.5789473684210527,
 0.6578947368421053,
 0.5526315789473685,
 0.5263157894736842,
 0.6052631578947368,
 0.5789473684210527]
```

- Dimension 5

5 principal component(s)




```
[0.5789473684210527,
 0.5526315789473685,
 0.5,
 0.6578947368421053,
 0.5789473684210527,
 0.5789473684210527,
 0.5789473684210527,
 0.6578947368421053,
 0.5789473684210527,
 0.6578947368421053]
```