Principal Component Analysis of the Breast Cancer Data Set

Fabio Setti

2023-05-08

Introduction

The following project focuses on the implementation of principal component analysis (PCA) on the Breast Cancer Wisconsin (BCW) data set. This data set presents 546 observations, each having 30 different numeric features generated from fine needle aspiration of breast masses. Most of the recorded features describe properties of breast cell nuclei such as radius, texture, perimeter, compactness and so on. Additionally, each observation is categorized as either malignant (breast cancer positive) or benign (breast cancer negative). The project report will have 2 parts:

- **Part 1: Algorithm Implementation.** Mathematics behind the PCA algorithm implementation.
- **Part 2: Tests and Results.** Component extraction. Effect of different λ parameters. Evaluation of how well the extracted components can distinguish between *malignant* vs *benign* breast cancer compared to the original 30 features in the BCW data set.

Part 1: Algorithm Implementation

Minimizing Reconstruction Error (RE)

The goal of PCA is to find a set of features smaller than the original feature space such that the reconstruction error (RE) between the original observations and the reconstructed observations calculated by projecting the original observations onto the lower feature space (i.e., the principal components) is minimized. The PCA algorithm that will be implemented will learn principal components through the minimization of the following equation:

$$RE = \sum_i \sum_j (x_i^j - (u^{q^T} x_i)u^q)^2 \tag{1}$$

Where x_i represent observation i, x_i^j represents the value of observation i for feature j, and u^q , with possible values of q ranging from 1 to j_{max}, represents the q_{th} principal component vector with j_{max} elements. Essentially, the goal is to find the vector q that minimizes the reconstruction difference between the original observations, x_i^j, and the observations reconstructed by projecting x_i onto the principal component u^q, which is done through the $(u^{q^T} x^i)u^q$ term in equation 1.

To achieve this goal, an iterative algorithm such as gradient descent needs to be implemented. As such, the first step is to calculate the partial derivative of equation 1 with respect to u^q. However, given that u^q is a vector, to simplify the gradient calculation, it is useful to restate equation 1 by just including scalar terms. Namely, it is possible to restate the $(u^{qT}x^i)u^q$ term through the following general notation:

$$(u^{qT}x_i)u^q = \sum_i \sum_j (u_j^q x_i^j)u_j^q \tag{2}$$

where u_j^q is a scalar term. Crucially, it is possible to update single elements of vector u^q in iterative fashion by performing gradient descent i times for each u_j^q sequentially. To this end, it is necessary to calculate the partial derivative of equation 2 with respect to u_j^q for each observation. To illustrate the concept, let us assume that we want to reconstruct a single observation with 2 features; the RE formula that needs to be minimized to find the first element of u^q, u_1^q, would be:

$$RE_{x_1} = (x_1 - (u_1^q x_1 + u_2^q x_2)u_1^q)^2 \tag{3}$$

Similarly, the RE formula that needs to be minimized to find the second element of u^q, u_2^q, would be:

$$RE_{x_2} = (x_2 - (u_1^q x_1 + u_2^q x_2)u_2^q)^2 \tag{4}$$

Equations 3 and 4 are a simplified notation of equation 1 that just restate the whole equation without vectors or summations to make the derivation simpler. Now, it is trivial to find the first derivatives with respect to the desired u_j^q element of u^q, which, in the case of equation 3 and 4, are respectively (the superscript q is omitted for simplicity):

$$\frac{\partial RE_{x_1}}{\partial u_1} = 2(-2x_1u_1 - x_2u_2)(x_1 - u_1(x_1u_1 + x_2u_2)) \tag{5}$$

$$\frac{\partial RE_{x_2}}{\partial u_2} = 2(-2x_2u_2 - x_1u_1)(x_2 - u_2(x_1u_1 + x_2u_2)) \tag{6}$$

Where it should be easy to see the pattern of how this formula generalizes when further features are added to the observations. The steps for the derivation of equation 3 (which is parallel to that of equation 4) can be found in the Appendix.

L_2 regularization

As an improvement to the given PCA algorithm (i.e., equation 1), I will implement L_2 regularization by including an additive term, λu^2, to the classical RE equation. Thus, the full equation to minimize will be:

$$RE_{L_2} = \sum_i \sum_j (x_i^j - (u^{qT} x_i) u^q)^2 + \lambda u^2 \tag{7}$$

Given that the L_2 regularization term is additive, the calculation of the gradient for equation 7 is essentially the same as equation 5 and 6, where the u in the regularization term is the element of u^q that the gradient is being calculated with respect to. For instance, the equivalent of equation 5 with L_2 regularization is simply:

$$\frac{\partial RE_{x_1}}{\partial u_1} = 2(-2x_1 u_1 - x_2 u_2)(x_1 - u_1(x_1 u_1 + x_2 u_2)) + 2\lambda u_1 \tag{8}$$

The term λ represents an added penalty that controls the size of the individual feature-weights within each component. In practice, the larger the λs, the smaller the overall feature-weights will be. L_2 regularization should prevent overfitting that may occur by extracting components that describe too closely the training data and do not generalize well to new data points. Additionally, it can be seen that when $\lambda = 0$, no regularization will occur and normal PCA as per equation 1 will be performed.

Iterative Extraction of Components

The PCA function implemented in this project, as well as the PCA process more generally, involves 2 general recursive steps:

Step 1: In step 1, a set of data points with j features is given as input; gradient descent according to the general form of equation 8 for each of the elements of a u^q component is performed. For this project all of the weights of u^q are initialized at 1 and progressively updated as gradient descent takes place. After that, the u^q vector that best minimizes equation 7 given the input data is returned.

Step 2: The data is reconstructed and is subtracted from the original input data:

$$X_{new} = X - \tilde{X} \tag{9}$$

Where X represents the original input data, and $\tilde{X}$ represents the reconstructed data. It is useful to note that equation 9 is equivalent to equation 1, since $X = x_i^j$ and $\tilde{X} = (u^{qT} x^i) u^q$. Then, X_{new} is fed back into the PCA function in *step 1* and a new component u^{q+1} that minimizes equation 7 given X_{new} will be extracted.

This process can be repeated up to the total number of features in the original data set.

Optimal Number of Components

The cardinal assumption of PCA is that the original feature-space can be *closely* approximated by a smaller feature-space. Indeed, the goal is to *closely* reconstruct the original feature space, not perfectly. However, since it is possible to extract as many components as original features, it is beneficial to define what the *optimal* number of components to be extracted is. Although there are many ways of defining the optimal number of components, most of them revolve around the notion of minimizing the difference between the original data and the reconstructed data (equation 9), while weighing the benefit of extracting an additional u^{q+1} component.

In the case of this project, the quantity that will be calculated to determine the benefit of extracting a u^{q+1} component will be the difference between 1 and the ratio between the RE of component u^q and the RE of component u^{q+1}. More practically, this quantity can be interpreted as the loss of magnitude of RE reduction as more components are extracted:

$$\Delta RE = 1 - \frac{RE_{u^{q+1}}}{RE_{u^q}} \tag{10}$$

Where it can be seen that when the RE for u^{q+1} is very similar to that of u^q, meaning that extracting u^{q+1} will produce negligible improvements in RE, ΔRE will approximate 0. This rule ought to be appropriate, as the RE improvement monotonically decreases as more components are extracted. This is the case because finding the vector u^q that best minimizes equation 7 means finding the largest component given the data; once that component is taken out of the data, the next component will necessarily produce a smaller RE improvement than the previous one. Although there is no set in stone rule for what value of ΔRE represents a *negligible improvement*, it seems reasonable to set that rule somewhere at $\Delta RE = .05$ as a general heuristic. Ultimately, equation 10 ensures that the number of extracted components remain proportional to their practical usefulness.

Part 2: Tests and Results

In this part of the project report, the PCA algorithm will be implemented. Before moving on to the results, I would like to mention that the learning rate for the gradient descent of the PCA algorithm will be fixed to $\epsilon = 0.1/N_{datapoints}$. The rationale behind this choice comes from the notion that every element of component u^q is updated $N_{datapoints}$ times per iteration. Dividing the learning rate by the number of updates should ensure that the step size taken while performing gradient descent will be adjusted according to the sample size. In theory, this should mean that more data will afford more accurate but slower gradient descent (given higher number of total steps), while less data should result in slightly less accurate but faster gradient descent (to make up for the loss of total steps).

Effect of Iterations and λ Parameters

Before determining the optimal number of components and using the extracted components for prediction, it is important to test how different numbers of iterations and different λ parameters influence the PCA algorithm. Namely, this is to determine the conditions under which the algorithm performs better. To this end, Table 1 presents the ΔREs up to 10 components for 2 different iteration conditions (10 iterations, 20 iterations) $\times$ 4 different λ values (0, .1, .5 , 1). The component extraction was performed on a random 75% split of the BCW data set.

Table 1: ΔRE**s for Iterations** $\times$ λ**s**

	$\lambda = 0$		$\lambda = 0.1$		$\lambda = 0.5$		$\lambda = 1$	
	Itr = 10	Itr = 20	Itr = 10	Itr = 20	Itr = 10	Itr = 20	Itr = 10	Itr = 20
u1	0.4269	0.4269	0.4261	0.4261	0.4224	0.4224	0.4163	0.4163
u2	0.2729	0.2729	0.2762	0.2762	0.2874	0.2874	0.2963	0.2963
u3	0.1020	0.1020	0.1101	0.1101	0.1375	0.1375	0.1564	0.1564
u4	0.0550	0.0522	0.0669	0.0646	0.0574	0.0831	0.0358	0.0428
u5	0.0297	0.0551	0.0282	0.0546	0.0500	0.0415	0.0095	0.0155
u6	0.0853	0.0570	0.0888	0.0610	0.0236	0.0389	0.0013	0.0015
u7	0.0201	0.0226	0.0076	0.0128	0.0000	0.0004	0.0007	0.0009
u8	-0.0003	0.0111	-0.0008	0.0023	-0.0002	0.0003	0.0004	0.0006
u9	-0.0009	0.0006	-0.0012	0.0012	-0.0002	0.0002	0.0003	0.0004
u10	-0.0010	0.0003	-0.0013	0.0008	-0.0003	0.0002	0.0002	0.0003

[a] Itr = number of total iterations where at each iteration $N_{observations} \times N_{features}$ inner iterations are performed (run time 10 Itr $\sim$ 12 sec., run time 20 Itr $\sim$ 24 sec.)

[b] Each row represents the ΔRE produced by component u^n

Although it may not be readily apparent, I believe that Table 1 gives some valuable insight into the algorithm's behavior. For instance, the ΔRE values should, ideally, monotonically decrease as more components are extracted. However, it can be seen that this is not necessarily the case for certain columns: the ΔRE_s tend to go back up after the u^6 component is extracted in all 10 iteration instances except for when $\lambda = 1$. Additionally, it also appears that, in the case of 10 iterations (except for $\lambda = 1$), the ΔRE_s sometimes become negative (although still really close to 0), around the extraction of the u^8 component. Clearly, this should not happen, as this implies that further component extraction produces higher RE. Still, this pattern is by no means concerning, but rather only informative, as I believe that it points to the role of number of iterations and λ values in algorithm convergence and how it gets progressively harder to extract components that produce small RE improvements.

In the case of iterations, it appears that a fairly lower number of iterations is necessary for convergence when extracting larger components (i.e., that reduce most of the RE) such as u^1 to u^3, whereas a higher number of iterations is needed to appropriately extract smaller components (i.e., that produce minimal RE reduction) such as u^6 and on. This is evidenced by the fact that the 20 iteration conditions consistently show a monotonically decreasing pattern of ΔRE_s. Simply put, larger components are more apparent and take less time to identify, whereas smaller components are harder to tell apart and take longer to identify. Consistent with this notion, it can be seen that differences in ΔRE_s between 10 and 20 iterations only start appearing when u^4 is extracted.

The pattern produced by the different λ values is also quite interesting. First of all, it appears that higher λ values help with algorithm stabilization and convergence. The main source of evidence for this observation is that even with 10 iterations, when $\lambda = 1$, the pattern of ΔRE_s is monotonically decreasing. Essentially, it seems that higher λ values help with increasing algorithm efficiency for components that are harder to extract. Finally, Table 1 also shows the effect of L_2 regularization, as the individual improvement in RE across components becomes more spread out as λ increases: u^1 is larger when $\lambda = 0$ compared to u^1 when $\lambda = 1$; however, this pattern is reversed for u^2 and u^3, as they are larger when $\lambda = 1$ than when $\lambda = 0$.

Ultimately, it is possible to draw some key conclusions from Table 1 that can inform the choices made in the remainder of the project:

- Overall, the PCA algorithm seems to perform reasonably well when large components are being extracted, regardless of number of iterations and λ values. Minor inconsistencies start to occur where further components are arguably small. However, it has to be noted that, in practice, small components are generally not very useful.
- The addition of L_2 regularization appears to help with convergence and extracting more balanced components. The effects of λ on prediction are tested later in the paper. Although this may not be the case for the BCW data set specifically, the fine tuning of λ values and iteration may greatly help in convergence and component extraction.
- Depending on how conservative/liberal one wants to be with component extraction, 3 to 6 major components seem to emerge from the 30 features of the BCW data set.

Predicting Malignant vs Benign Breast Tumors

In this section, I will test how well the extracted components can predict malignant/benign breast tumors compared to the original 30 features of the BCW data set. Given that the outcome is binary, the prediction will be done through a logistic classifier. In all instances, the logistic classifier will be trained on a 75% split of the original data set, and the trained classifier will be tested on the remaining 25% split of the data set.

The measure of performance that will be used for the classifier will be accuracy, $Accuracy = \frac{Y_{correct}}{Y_{total}}$, where $Y_{correct}$ represents the number of correctly classified data points and Y_{total} represents the total number of data points to classify. I would argue that accuracy is an appropriate measure in the case of the BCW data set, because the two possible outcomes are fairly evenly distributed (benign $\sim$ 63% ; malignant $\sim$ 38%).

Given what has been learned from the results in Table 1, I will extract 6 components (10 maximum iterations) under 3 different λ values, 0, .5, and 1. I will not extract components with $\lambda = .1$, as it seems that, judging by Table 1, the difference between $\lambda = 0$ and $\lambda = .1$ is negligible. Although 10 iterations turned out to produce somewhat inconsistent components when $\lambda = 0$, I suspect that the impact on prediction accuracy will be minimal. Additionally for each of the 3 λ conditions, I will extract components from an 80%, 60%, and 40% split of the original data set. This last step is undertaken to explore how the relative number of data points used to learn components influences prediction. Essentially, should accuracy be similar between components learned from differently sized splits, that would indicate that the PCA algorithm can learn components that can accurately summarize the original data from a small number of data points.

Finally, accuracy for all $\lambda_s \times \%_{splits}$ conditions will be calculated progressively for each subset of components as predictors (i.e., just 1 component up to all 6 components). Since the accuracy of the full 30 features should theoretically always exceed that of the 6 components, rather than the accuracy of the components themselves, I will present the difference in accuracy, $\Delta Accuracy = Accuracy_{30features} - Accuracy_{components}$. Thus, smaller $\Delta Accuracy$ implies closer prediction accuracy between the 30 features and the components. The *Accuracy* of the logistic classifier with 30 features was .98 (98% accuracy), which indicates that the data is quite reliably separable. Results for $\Delta Accuracy_s$ are displayed in Table 2.

Table 2: $\Delta accuracies$ **for** $\lambda_s \times \%_{splits}$ **Conditions and Varying Components**

	$\lambda = 0$			$\lambda = 0.5$			$\lambda = 1$		
	80%	60%	40%	80%	60%	40%	80%	60%	40%
u1	0.063	0.056	0.063	0.063	0.056	0.063	0.063	0.056	0.063
u1...u2	0.056	0.063	0.056	0.056	0.063	0.056	0.056	0.063	0.056
u1...u3	0.070	0.063	0.056	0.070	0.063	0.056	0.070	0.063	0.056
u1...u4	0.014	0.014	0.049	0.014	0.014	0.056	0.014	0.014	0.042
u1...u5	0.007	0.014	0.007	0.007	0.014	0.014	0.014	0.014	0.035
u1...u6	0.007	0.007	0.007	0.007	0.014	0.007	0.014	0.014	0.035

[a] The numbers displayed above are the difference beetween the 30 features classifer (.98) and the accuracy given the cell condition

Once again, a great deal of information can be extracted from Table 2. Probably, the most important result is that with just a single component (row 1), the accuracy is already quite close to the accuracy of the 30 original features, with $\Delta Accuracy_s$ being either .063 or .056. This seems to suggest that the first component extracted very closely approximates the relation between the outcome and the 30 features.

Before any further discussion about accuracy, it would be useful to make a general consideration about the overall pattern of $\Delta Accuracy_s$ in relation to the 9 different conditions ($\lambda_s \times \%_{splits}$). By looking at Table 2, it can quickly be seen that most rows present the same $\Delta Accuracy_s$, regardless of condition. Actually, I would guess that most of the variation that happens (if at all) is due to the random state chosen to split the data during components learning. Indeed, there is essentially no noticeable difference (within row $\Delta Accuracy_s$ differences is never $> .05$) in $\Delta Accuracy_s$ from one condition to another.

There are many possible explanations for this finding, but my best guess is that this has to do with the nature of the data set. Namely, it seems to me that there is a set of highly correlated features that is strongly related to the outcome, and this is what the first component is picking up. After the first component is extracted, not much of the remaining data is related to the outcome, and further addition of components only produces minimal improvements in prediction. This notion is further exemplified by some of the $\Delta Accuracy_s$ seen in rows 2 and 3; in these rows, it is shown that prediction accuracy barely improves (row 2), or even slightly decreases (row 3), when the logistic classifier is trained on the additional components 2 and 3. In particular, it seems that feeding component 3 as a feature to train on to the classifier merely makes the prediction task harder. At first glance this may seem odd, given that both component 2 and 3 produced a large improvement in RE in Table 1; however, this result is in line with my suspicion that only certain features in the original data are related to the outcome. Namely, components 2 and 3 reconstruct a significant portion of the original data, but that reconstructed portion is probably largely unrelated to the probability of a breast tumor being malignant or benign. It is not until component 4 (row 4), which only reconstructs a small portion of the original data, is fed to the logistic classifier that accuracy shows a reasonable, although still small, improvement. Finally, components 5 and 6 produce virtually no change in accuracy.

Consistent with the results in Table 1, although the extracted components are somewhat different depending on the λ values, the results are largely consistent. Again, this is probably due to how large and easy to extract component 1 is and to how quickly early components tend to converge. It is entirely possible that different λ values will result in noticeably different outcomes for other data sets.

Conclusion

The implementation of the PCA algorithm turned out to perform quite well. Regardless of λ values and iterations, the algorithm seemed to be able to extract large components quite effectively. Finer tuning seemed to be needed to accurately extract smaller components, and in such cases, a higher number of iterations and λ values of either .5 or 1 seemed to help stabilize the algorithm. Still, as I mentioned, smaller components should be expected to be harder to extract and probably have limited practical usefulness.

From a prediction point of view, the extracted components also performed surprisingly well. Just a single component was good enough to reach a prediction accuracy of above 90%. This prediction accuracy was reached with a relatively small number of iterations for component extraction and regardless of λ values and % of the original data set used to learn components. Additionally, marginal prediction improvement was provided by extracting further components. As previously discussed, this is most likely due to the nature of the data set.

Appendix

Partial derivation of equation 3 with respect to u_1 with L_2 Regularization

$$\begin{aligned}
RE_{x_1} &= (x_1 - (u_1x_1 + u_2x_2)u_1)^2 + \lambda u_1^2 \\
\frac{\partial RE_{x_1}}{\partial u_1} &[(x_1 - (u_1^q x_1 + u_2x_2)u_1)^2 + \lambda u_1^2] \\
&= 2(x_1 - (u_1x_1 + u_2x_2)u_1) \times \frac{\partial}{\partial u_1}[x_1 - (u_1x_1 + u_2x_2)u_1] + 2\lambda u_1 \\
&= 2(x_1 - (u_1x_1 + u_2x_2)u_1) \times (\frac{\partial}{\partial u_1}[-(u_1x_1 + u_2x_2)u_1]) + 2\lambda u_1 \\
&= 2(x_1 - (u_1x_1 + u_2x_2)u_1) \times (-(u_1x_1 + u_2x_2) - (x_1 + 0)u_1) + 2\lambda u_1 \\
&= 2(x_1 - (u_1x_1 + u_2x_2)u_1) \times (-u_1x_1 - u_2x_2 - u_1x_1) + 2\lambda u_1 \\
&= 2(-2u_1x_1 - u_2x_2)(x_1 - u_1(u_1x_1 + u_2x_2)) + 2\lambda u_1
\end{aligned}$$

The final equation shown in teh previous line is equation 8 shown on page 3.