COURSEWORK

In this report I will demonstrate an implementation of the LDA dimension reduction technique on 2 datasets. The first one is two classes of bivariate gaussians, projecting the data on an optimal 1D space and the second is the Fisher's Iris dataset projecting the data from 4D to a 2D subspace. In the last part I will perform a non-linear regression using a polynomial function as basis model trying to fit sinusoidal form of data. All the images come from the jupyter notebook file I supplied.[1]

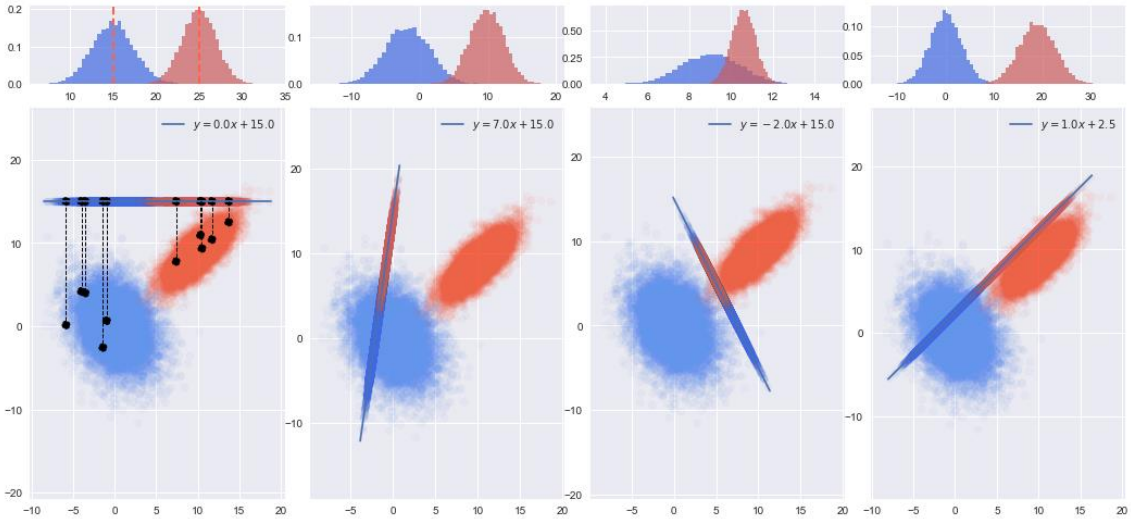
4. CLASSIFICATION

4.1. Data I: separate 2 Gaussians

1a.)

Starting with, every data point $P(x_p, y_p)$ is treated as a vector starting from (0,0) and pointing to (x_p, y_p), so its projection to the line $f(x) = ax + b$ can be achieved by taking the dot product of the point and the unit vector of the line where $\hat{w} = \frac{(x_1, f(x_1))}{\sqrt{x_1^2 + f(x_1)^2}}$. Thus, $P_{proj} =< P|\hat{w} >$.

The projections aren't subject to the intercept of the line as long as the slope remains the same. For convenience we treat the line as it comes through the origin and then move the projected points back to the original through solving algebraic the system. The result can be interpreted in 2D by multiplying it with the $\hat{w}$ of the line.



[1] In order to run the notebook file, run the whole file simultaneously. I tried to make the file as robust it could be, but this doesn't guarantee that will work for big modifications. Read the instructions in the end of the report.

1b.)

To calculate the Fisher score (*or better Fisher ratio*) we use the vectorized formula:

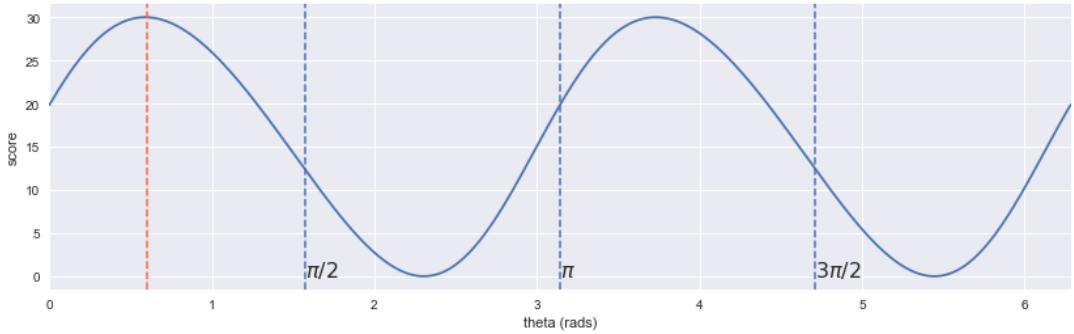
$$F(w) = \frac{w^T S_B w}{w^T S_W w}(n_a + n_b)$$

$$S_B = (\mu_a + \mu_b) \qquad S_W = n_a S_a + n_b S_b$$

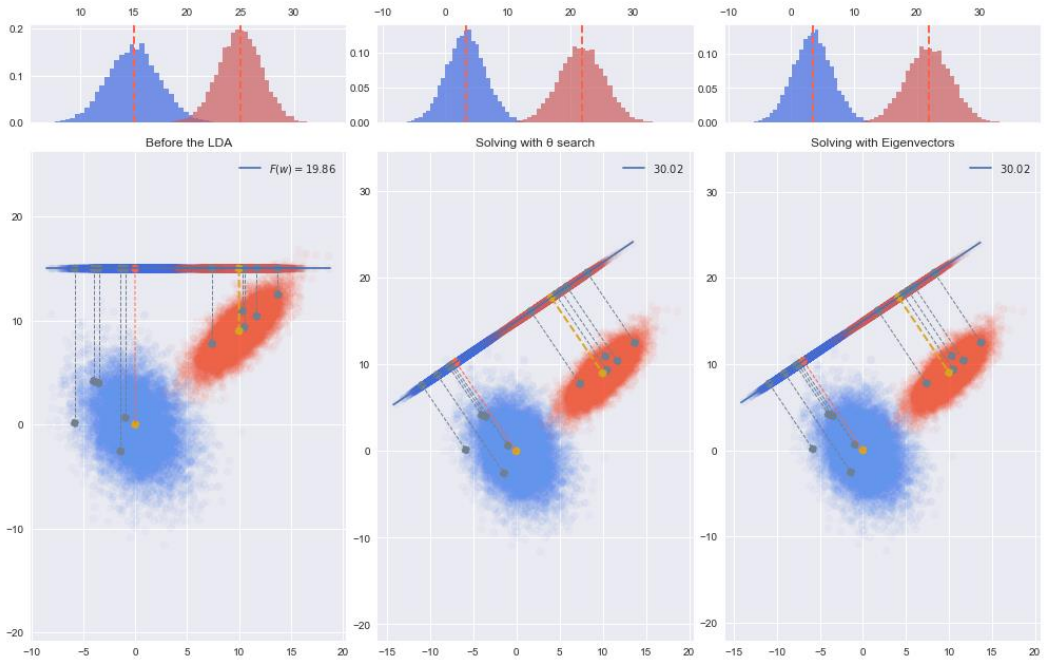
$$S_c = (x^n - \mu_c)(x^n - \mu_c)^T \qquad \mu_c = \frac{1}{n}\sum x^n$$

Then we rotate the line to find the degrees for which the score is maximized:

$w(\theta) = R(\theta)w_0 = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}\begin{pmatrix} x_w \\ y_w \end{pmatrix}$, For w_0, I used the line $y = 15$ which means $\hat{w}_0 = (1,0)$.



$argmax(F(w(\theta))) = 34.19°$ from the **initial position,** which corresponds to $\hat{w} = (0.83,0.56)$ and $F_{max}(w) = 30.02$.



It is clear from the plots that when the $\hat{w}$ points that direction, Fisher score gets maximized and the projected points have the maximum separation.

2a.)

To draw the equi-probable contour lines we first need to match every point with its probability which occurs from its position. This is easy with the .pdf function in python which basically calculates:

$$p(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^p|\Sigma|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

2b.)

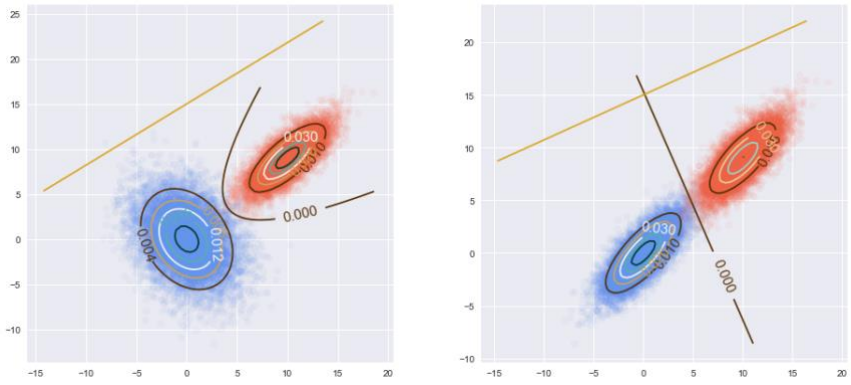
Log-odds are calculated by $\ln\left(\frac{P(c=a|x^n)}{P(c=b|x^n)}\right)$ where $P(c=a|x^n)$ indicates the probability of the x^n belonging to that particular class, thus log-odds equals zero acts as a decision boundary and can be calculated by the Bayes rule:

$$\ln\left(\frac{P(x^n|C=a)P(C=a)}{P(x^n|C=b)P(C=b)}\right) = g_a(x) - g_b(x) = f(x;\theta) + \theta_0$$

where

$$g_c(x) = lnP(C=c) + \frac{1}{2}\ln|\Sigma_c^{-1}| - \frac{1}{2}(x^T\Sigma_c^{-1}x - 2\mu_c^T\Sigma_c^{-1}x + \mu_c^T\Sigma_c^{-1}\mu_c)$$

It's easy to deduct from these formulas that when $\Sigma_a \neq \Sigma_b$, decision boundary has a quadratic form since we can't eliminate $x^T\Sigma_c x$ terms. The $\mu_c^T\Sigma_c\mu_c$ and $\ln(P(C=c)\sqrt{|\Sigma_c^{-1}|})$ act as independent terms. However, when $\Sigma_a = \Sigma_b$, then $x^T\Sigma_c x$ terms get eliminated and the remainder is a linear equation with its slope is defined by the $\mu_c^T\Sigma_c^{-1}x$ term.



2c.)

It is an interesting case when we make use of the unbalanced formula of Fisher's score. In the case where $n_a = n_b = n$ the balanced score gets: $F_{bal}(w) = \frac{w^T S_B w}{(w^T S_W w)n} 2n = 2\frac{w^T S_B w}{w^T S_W w}$. Making use of the unbalanced score, instead of being the same, score gets: $F_{unb}(w) = \frac{w^T S_B w}{w^T S_W w} = \frac{F_{bal}(w)}{2}$. In order to keep the scale, I **modified the formula** to:

$$F_{unb}(w) = 2\frac{(m_\alpha - m_b)^2}{(\sigma_a^2 + \sigma_b^2)}$$

The above formula is the same formula as stated first but I introduced a scaling factor in order to get comparable results with the balanced formula.

Example 1:

For $\Sigma_a = \begin{pmatrix} 6 & -2 \\ -2 & 9 \end{pmatrix}, \mu_a = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, n_a = 10000$ and $\Sigma_b = \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}, \mu_b = \begin{pmatrix} 10 \\ 9 \end{pmatrix}, n_b = 10000$

$$\boldsymbol{F_{bal}(w) = F_{unb}(w) = 30.02}$$

For $n_a = 2000, n_b = 10000$:

$$\boldsymbol{F_{bal}(w) = 27.20, F_{unb}(w) = 30.06}$$

Example 2:

For $\Sigma_a = \Sigma_b = \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}, \mu_a = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mu_b = \begin{pmatrix} 10 \\ 9 \end{pmatrix}, n_a = n_b = 10000$

$$\boldsymbol{F_{bal}(w) = F_{unb}(w) = 26.15}$$

For $n_a = 2000, n_b = 10000$:

$$\boldsymbol{F_{bal}(w) = 26.19, F_{unb}(w) = 26.11}$$

Apparently, we can see that the balanced formula gives weights to different classes' size in order to concentrate in the separation of the projection of the biggest class. Specifically, the direction of the line tends to **align the biggest discriminant axis**.

Applying the Bayes rule, the above consequence is integrated in the term $\ln\left(\frac{P(c=a)}{P(c=b)}\right)$ since when $n_a = n_b$ is eliminated. So, the decision boundary which derives from Bayes rule, is always balanced.

4II. Data 2: Iris data

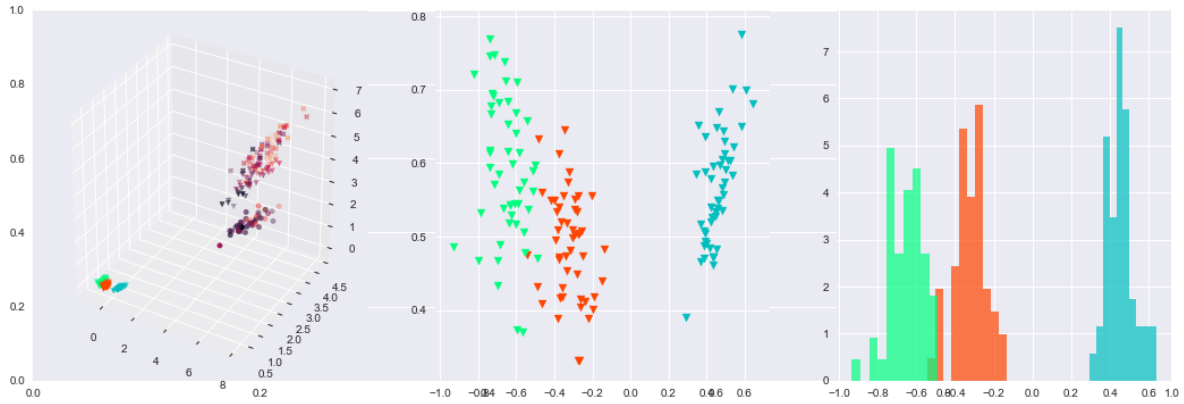
1.)

Finding the optimal value for the $F(w)$ needs only to solve $\nabla_w F(w) = \frac{\partial}{\partial w} F(w) = (S_B - \lambda S_W)w = 0$, where $S_B w = \lambda S_w w$ is the generalized eigenvalue problem and $\lambda = \frac{w^T S_B w}{w^T S_W w}$ the Rayleigh quotient or $\lambda(n_a + n_b) =$ Fisher's score and needs to be maximized. sp.linalg.eigh(SB,SW,type=1) solves the generalized eigenvalue problem. It is not recommended to solve directly $S_W^{-1} S_B w = \lambda w$ since S_W is not always invertible. Also, scatter-between and scatter-within class are Hermitian matrices since are covariance matrices, but this does not guarantee that $S_W^{-1} S_B w$ is Hermitian too.

S_B is constructed out of K classes (3 in our dataset) based on an outer product, which means that **its eigenvalues are K-1 the most**. In other words, the K centroids span at most a K-1 dim. space. (K should be less than the number of features p, otherwise if p<K-1 then the discriminant subspace is p-dimensional). In this dataset, the projection can't be achieved for more than 2 dimensions.

Intuitively, eigenvectors point the directions that does not get affected from the transformation $S_w^{-1}S_B$ (consider this matrix non-singular), so projecting on directions which this transformation shrinks them to zero, is not going to give us more information.

2.)



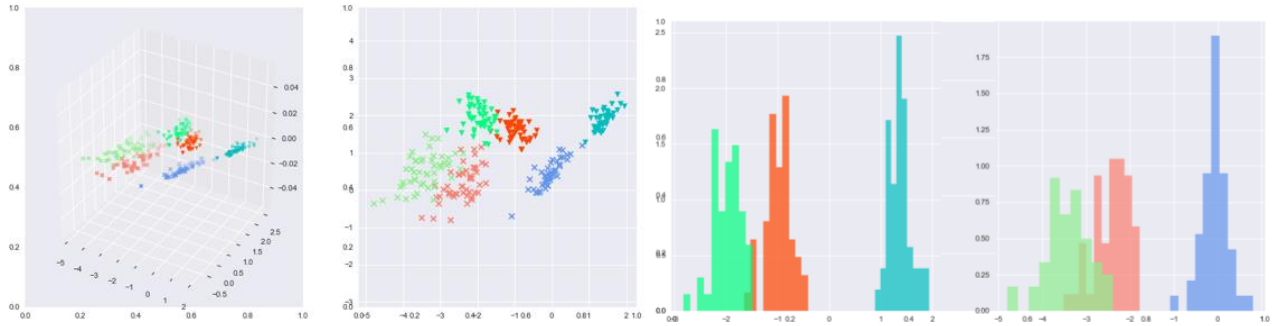
In the first plot, original data are plotted in 4D with color pointing the 4$^{\text{rth}}$D and the projected data lie in a 2D plane, subspace of original dimension space, which derives from the largest eigenvalues' eigenvectors of the generalized eigenvalue problem (in this case are 2. The other 2 eigenvalues were almost zero). The second plot shows a clear image of the projected points. Points merely overlap. The histograms show that even 1D subspace could project the data well separated (histogram of the other dimension doesn't provide anything useful that's why I skipped it).

The subspace of the original space needs 4D vectors to be constructed regardless its dimension. In our example, 2 vectors of 4D act as a base of the reduced dimension space.

3.)

Projecting the data on a subspace coming from $w = w^* + a$, where a comes from the rest of the eigenvectors will do nothing more than displacing the data in a common direction. We have added a linear combination of linear independent vectors. If eigenvectors were coming from a symmetrical $n \times n$ matrix they would be orthogonal and, in this case, would get displaced uniformly keeping their formation.

However, they are not deriving from a Hermitian transformation and as a result they are not orthogonal. Also, keep in mind that the rest 2 eigenvectors do not contain much useful information (because of their small eigenvalues) and thus, they act as a kernel of the transformation. This is the reason that the resulting transformation lies in the same plane (because of the addition of the null space). Also, note that projections seem squeezed. I cannot justify this last observation, but I will attribute it to the non-orthogonality of the vectors (without meaning this that I am correct).



You can see that projections moved along the rest 2D of the original space and look like their general formation is kept but their distributions are squeezed.

4.)

To summarize, LDA is a dimension reduction technique which tries to maximize the Rayleigh quotient. It is great when the number of classes is equal or less with the number of features. In the first example we saw how from a 2D vector space with 2 classes of 2 features we can project data to the 1D and classify them without losing much information. In the second example, we projected data from a 4D vector space with 3 classes of 4 features to a 2D subspace. The maximum dimensions of the projection subspaces come from the condition $\min(K-1, p)$ where K: classes and p: features.

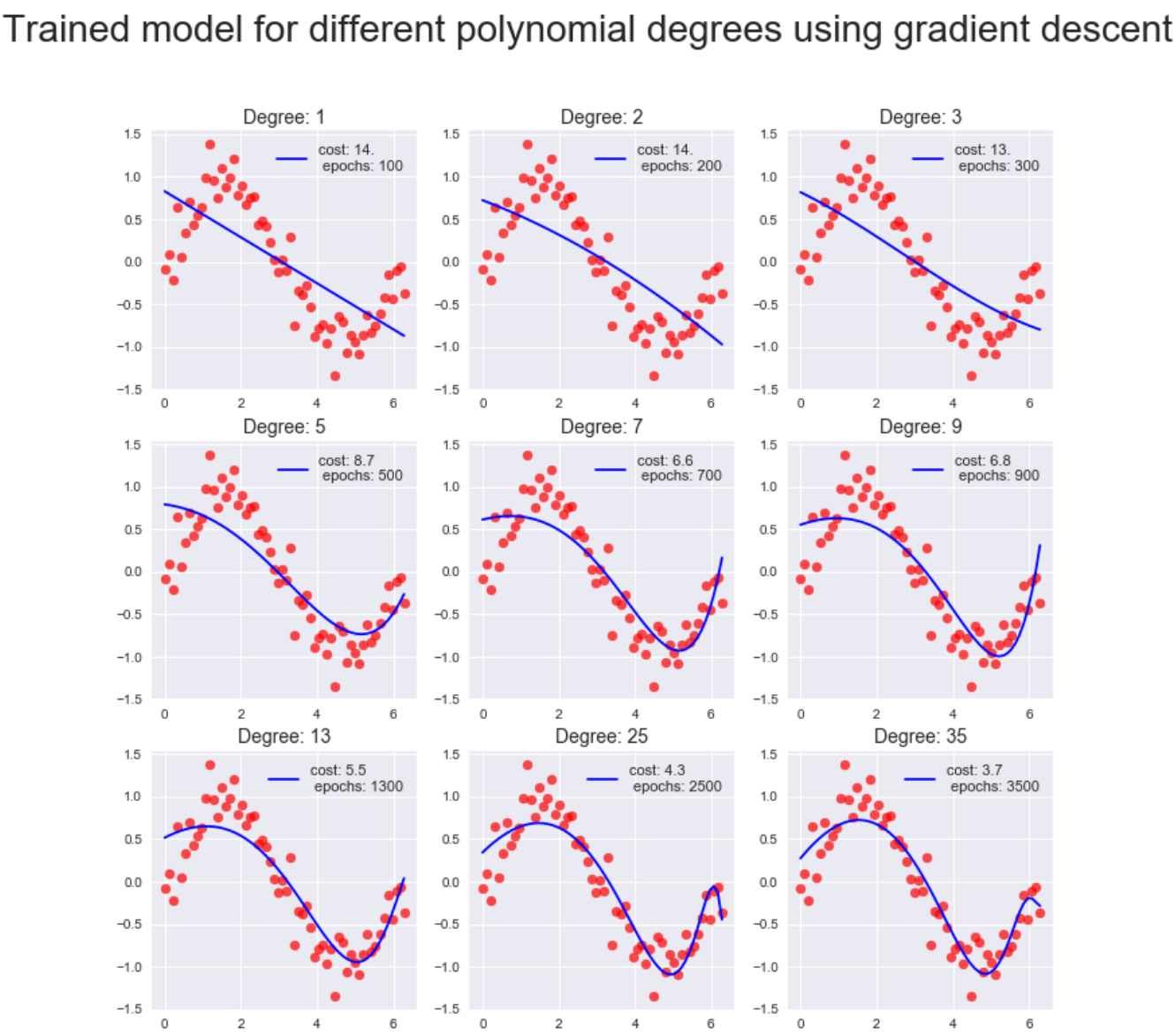
5. LINEAR REGRESSION WITH NON-LINEAR FUNCTIOS

5I. Performing linear regression

1.)

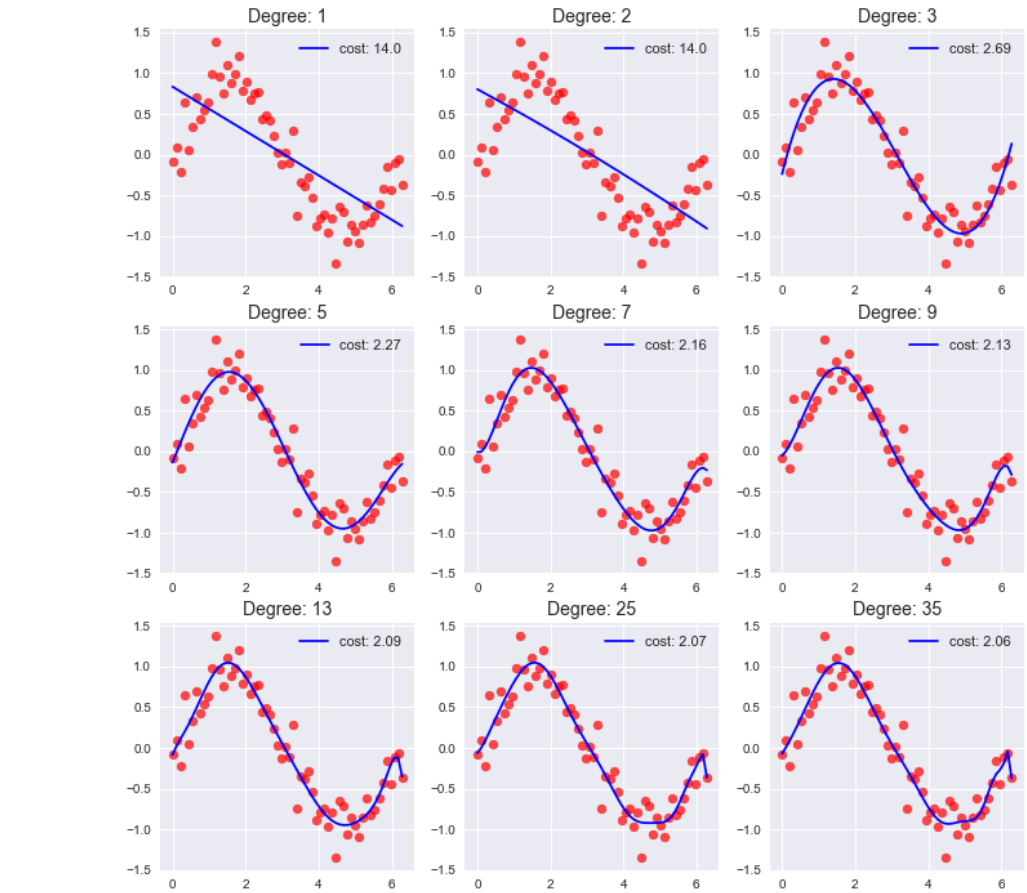
In this section I tried to fit a polynomial basis function on a sinusoidal wave form using gradient descent. I tried to obtain the basic polynomial: $\varphi(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ forming a design matrix with n-powers of x and tried to find the corresponding weights. In order to have all the weights the same scale I normalized the corresponding powers of x by $\frac{X-\mu}{\sigma}$.

My initial plan was to neglect the L2 norm to see how the polynomial function will overfit in the higher degrees but using the gradient descent algorithm in such complex data without setting a learning rate schedule, made extremely hard the training process in less than 5000 epochs. These are the results of the gradient descent for various polynomial degrees:

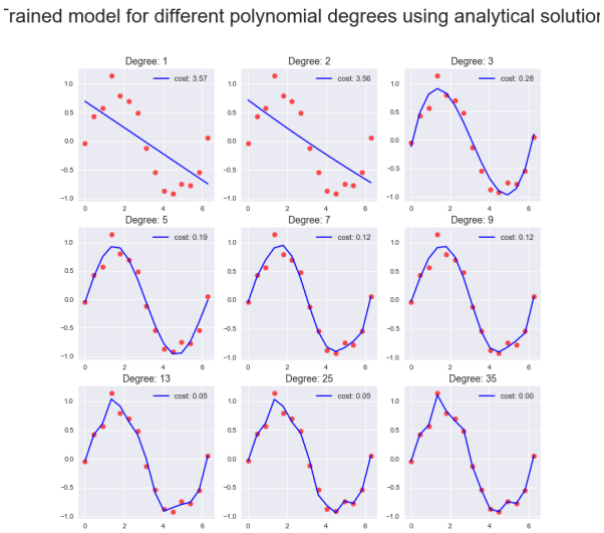


2.)

It can be seen that some overfit starts to occur with degrees over 25 but it wasn't satisfying; gradient descent couldn't do much without adjusting more hyperparameters. In contrast, solving analytically the cost function with the help of the pseudo inverse, again with insignificant regularization the overall fit was extremely better specially in lesser degrees:

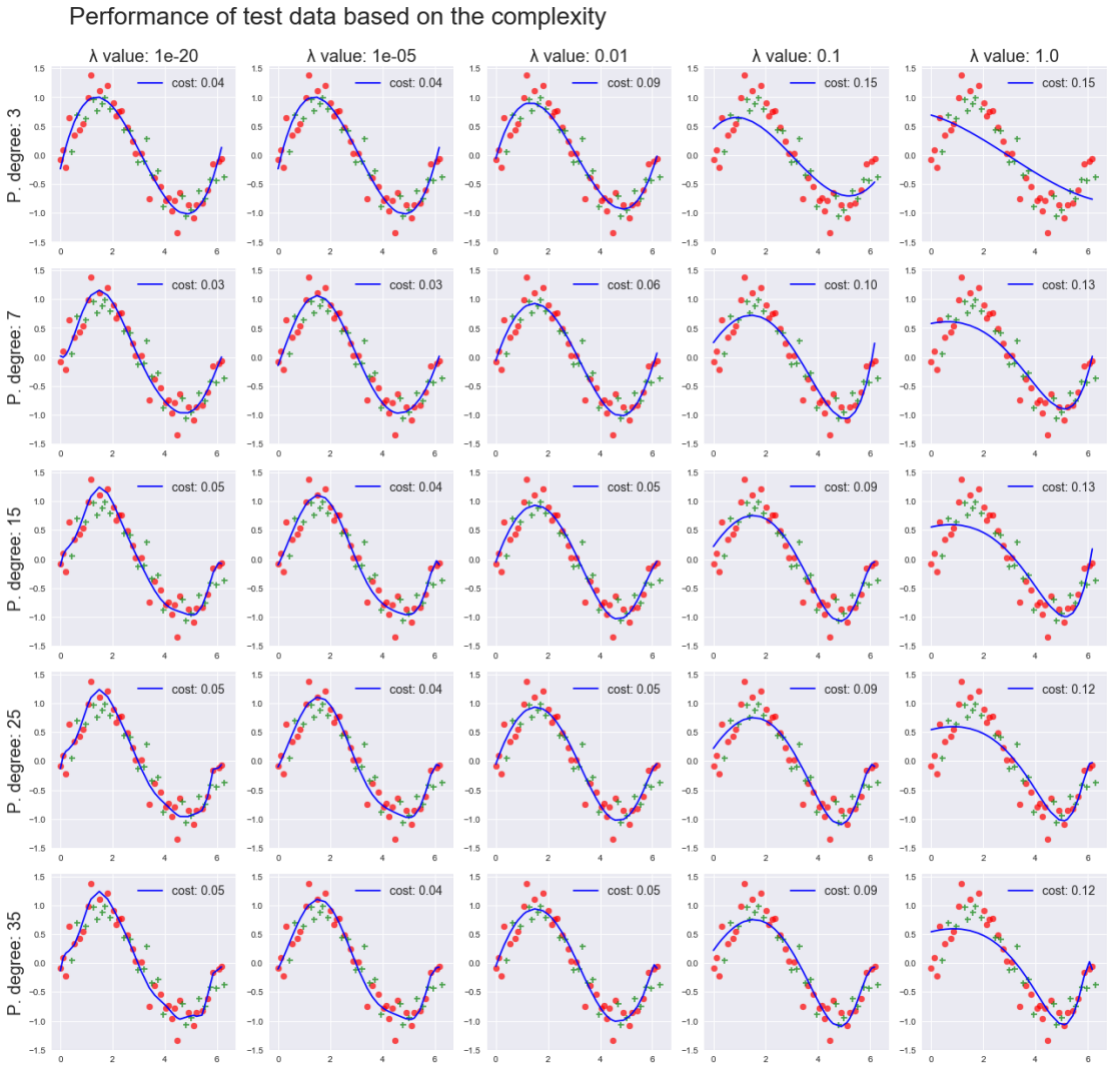


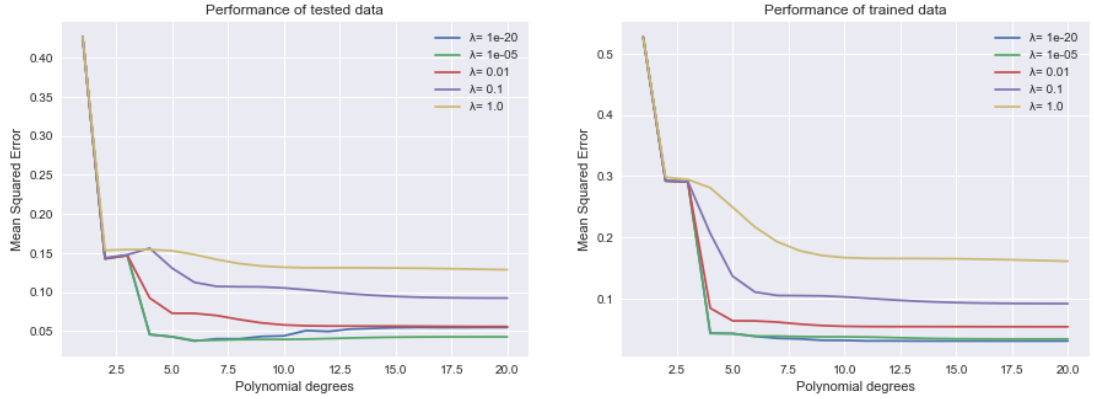
- The reason that overfitting is not that obvious is due to the big number of samples. Reducing the number of samples in the gradient descent case will not help the learning procedure but in the second case we can see:



3.)

In this part after seeing how the model reacts (underfit/overfit) to the number of samples, the strength of regularization, the polynomial degree, and on the method used to minimize the loss function, I will plot its performance based on some random test samples in function of the regularization term and the number of basis components $\varphi(\cdot)$.





5II. How does linear regression generalize?

1.)

In the last section, I split 50 sample data in 10 pieces in order to do 10-fold cross validation for the same parameter values (regularization strength and basis polynomial degree) as above:

```
Average error for 10-fold cross validation:
['-----' '1e-20' '1e-05' '0.01' '0.1' '1.0']
[[ 3.     0.748  0.749  0.936  1.167  0.988]
 [ 7.     0.736  0.774  0.845  0.918  1.02 ]
 [15.     1.028  0.779  0.81   0.926  1.009]
 [25.     2.869  0.782  0.815  0.916  1.002]
 [35.     1.987  0.782  0.816  0.914  0.984]]
```

It is clear that the error gets smaller for reasonable regularization strength as the polynomial function gets less complicated (less degrees). Also, error gets smaller for every degree as the regularization is between $\mathbf{10^{-5} - 10^{-2}}$**.** This means that **tested samples get higher accuracy for less overfitted models.**

INSTRUCTIONS TO RUN THE JUPYTER NOTEBOOK:

In the jupyter file you can modify $\Sigma_a, \Sigma_b, m_a, m_b, n_a, n_b$, the four projection lines w, the number of samples N in the non-linear regression part, the proportion between train/test data, the polynomial degree matrix and the regularization coefficient matrix.

The purpose of this report isn't to analyze the technical issues behind every implemented algorithm, and for that reason everything important is well readable in the script. Large coding cells are mostly for visual interpretations titled as "VISUALIZATION CELL" and you can ignore them.

Outputs present the results I demonstrated in this report, but if you want to modify some parts (in order to check for example, the linear/quadratic form of the decision boundary for various Σ_a, Σ_b run all the cells at once). Reproducibility is guaranteed through np.random.seed.

Bibliography

[1] Barber, D. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[2] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[3] Hastie, T., Tibshirani, R. and Friedman, J. H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[4] Lectures' slides