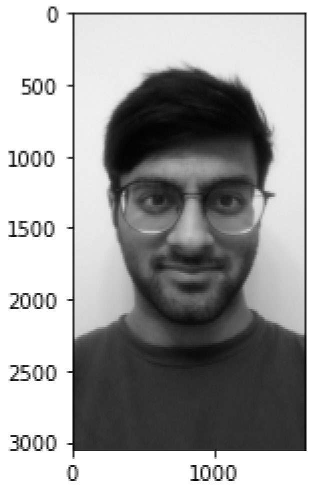
Math 499 Project Problems

Sai Thatigotla

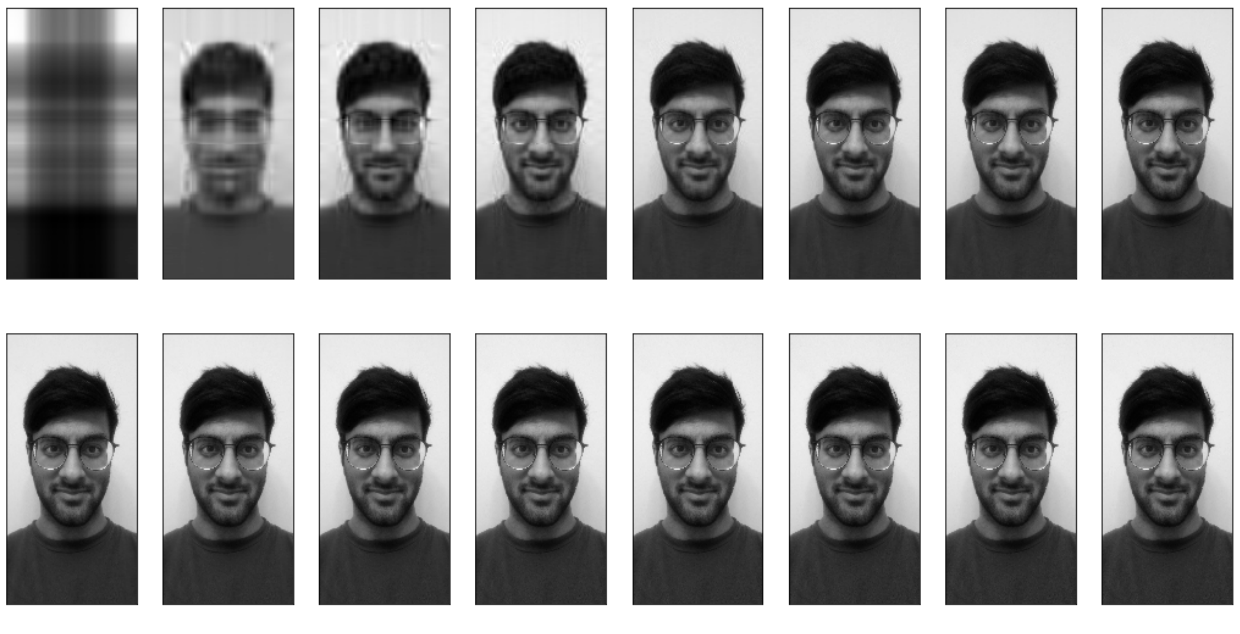
Problem 1

In this problem, we took a selfie and compressed it using Singular Vector Decomposition (SVD). SVD decomposes a matrix A (shape of m x n) into 3 matrices, U (m x m), a diagonal Sigma (m x n), and V Transpose (n x n). And since U and V are orthogonal matrices, U, and V correspond to rotations of the matrix and the diagonal Singular values to a scaling of the columns of U with the rows of V. And since the columns of U and the rows of V are sorted by how important they are in forming matrix A, we can reduce the amount of columns in A (from the back since the beginning is the most important one), then the amount of Singular values used to make the diagonal Sigma (from the back of the columns and rows as well) and then the rows of V to compress the image. By choosing less ranks of the matrix, we can then reconstruct an approximate version of A based on only some of the ranks. And since the matrices can be stored separately instead of being reconstructed, we can also save storage space since you only have U + S + V amount of bytes to keep.

The original image looks like the following with a shape of 3046 x 1632:

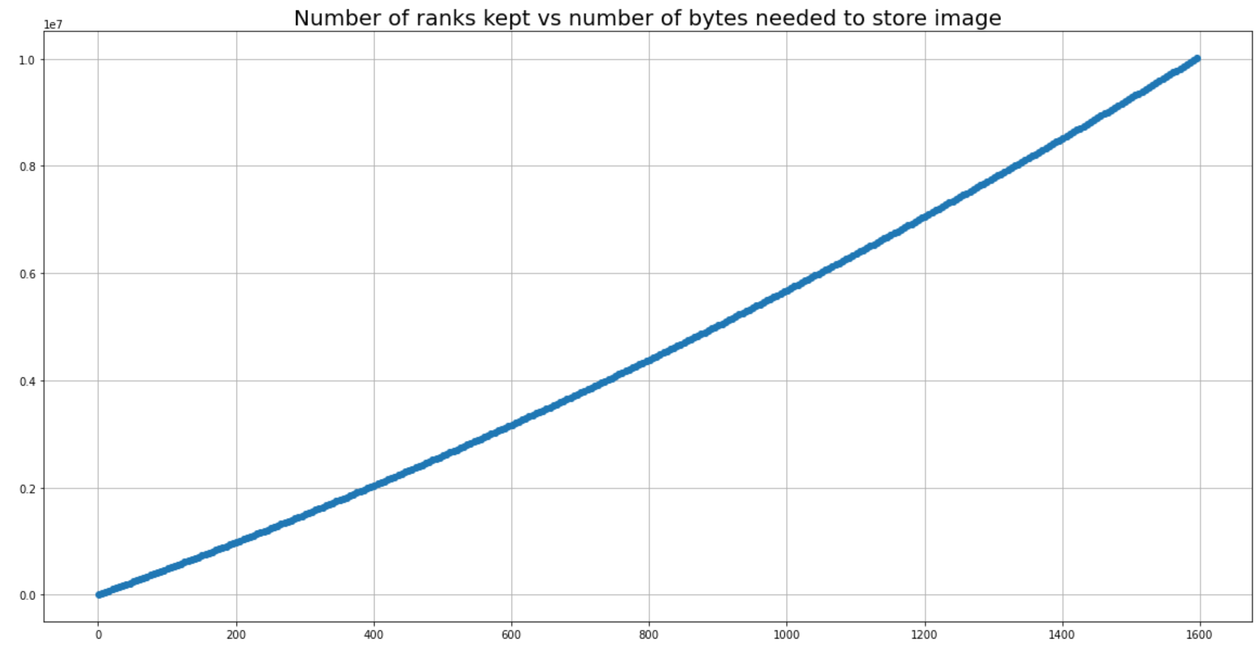


Then, from left to right and top to bottom the following image shows compression based on these ranks (1, 5, 10, 20, 40, 60, 80, 100, 150, 175, 200, 400, 500, 800, 1100, 1600):



As you can see, images get compressed and more blurry the less ranks you keep. However, you can also notice that the blurriness becomes much less noticeable after a couple of images. This may be due to the fact that the image is roughly symmetrical, and the background and my shirt are uniform colors. This can be kind of seen when you only keep the 1st rank since the most important columns and rows would have been roughly based on the center columns (since it kept the upper white corners).

If we want to know how many ranks we should keep to save on storage, we can plot the number of ranks kept vs the storage space needed and find an elbow/knee point in the curve, such as in the following:

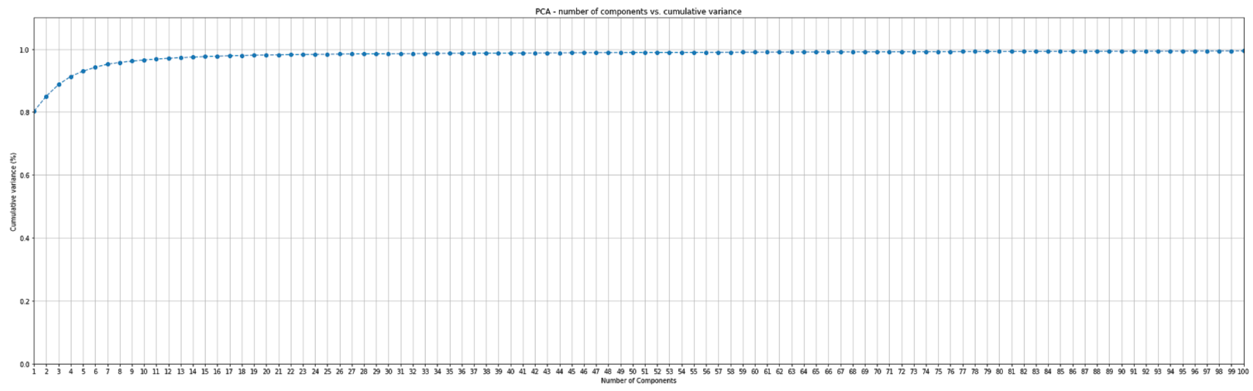


The elbow point obtained from using a package called kneed was 40 ranks (5th image in the first row) or 188720 bytes of space needed vs the original image storage of 4971072 bytes.

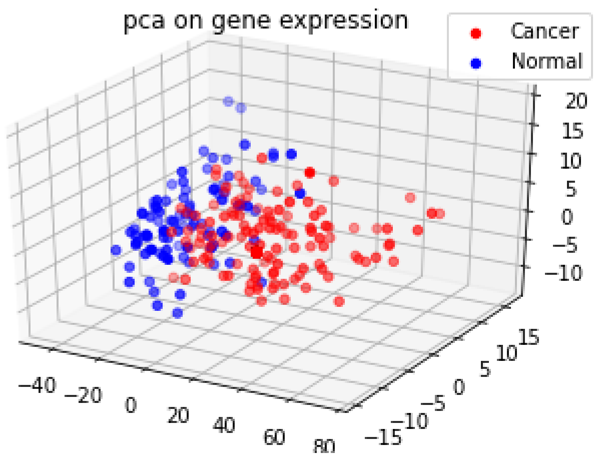
Problem 2

In this problem we used the ovarian cancer dataset to analyze principal component analysis (PCA). PCA is a linear transformation of the data where (using Eigenvalue decomposition) it creates a matrix where the columns of the new matrix are the eigenvectors of the covariance matrix from the original data. All of these eigenvectors are unit and orthogonal to each other. Geometrically, you are fitting an ellipsoid to the original data where each orthogonal axis corresponds to maximizing the variance of the data to the axis. Since you also have the eigenvalues when you get the eigenvectors, you can sort the eigenvalues by the amount of variance each axis represents by dividing the eigenvalue of an axis with the sum of all the eigenvalues.

So, if we decide to limit the amount of principal components we keep, we can get a lower dimensional representation of the data. This is useful because of the curse of dimensionality problem. If we want to find out how many components to keep, we can plot the number of components vs the cumulative variance such as in the following:



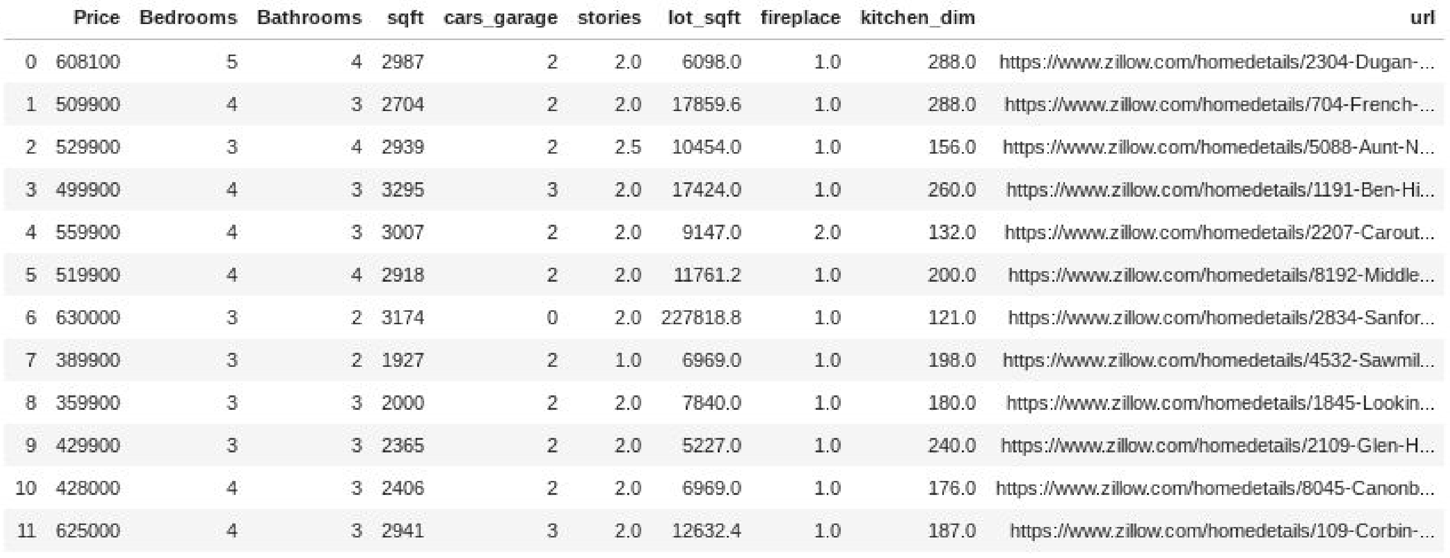
The image is kind of hard to see, but 3 components correspond to roughly 90% of the cumulative variance even though the original data had 15000 features. The projection of the first 3 principal components to the original data (a change of basis) is the following:



From here, we can see the 3 components were enough to create clusters of patients, though there was still some overlap. It might be difficult to get a good linear decision boundary, though it is difficult to tell without rotating the image or actually trying to classify.

Problem 3

In this problem we collected data on houses for sale in the same area (probably to account for differences between locations) and used LASSO regression with the coordinate descent algorithm to model the price. I chose the following features: Number of Bedrooms, Number of Bathrooms, square footage of house, numbers of cars in garage, number of stories, square footage of the lot, how many fireplaces, and the product of the kitchen dimensions. The data on 12 houses were all on sale in Nolensville, TN and the information was collected from Zillow. My data table is as follows:



	Price	Bedrooms	Bathrooms	sqft	cars_garage	stories	lot_sqft	fireplace	kitchen_dim	url
0	608100	5	4	2987	2	2.0	6098.0	1.0	288.0	https://www.zillow.com/homedetails/2304-Dugan-...
1	509900	4	3	2704	2	2.0	17859.6	1.0	288.0	https://www.zillow.com/homedetails/704-French-...
2	529900	3	4	2939	2	2.5	10454.0	1.0	156.0	https://www.zillow.com/homedetails/5088-Aunt-N...
3	499900	4	3	3295	3	2.0	17424.0	1.0	260.0	https://www.zillow.com/homedetails/1191-Ben-Hi...
4	559900	4	3	3007	2	2.0	9147.0	2.0	132.0	https://www.zillow.com/homedetails/2207-Carout...
5	519900	4	4	2918	2	2.0	11761.2	1.0	200.0	https://www.zillow.com/homedetails/8192-Middle...
6	630000	3	2	3174	0	2.0	227818.8	1.0	121.0	https://www.zillow.com/homedetails/2834-Sanfor...
7	389900	3	2	1927	2	1.0	6969.0	1.0	198.0	https://www.zillow.com/homedetails/4532-Sawmil...
8	359900	3	3	2000	2	2.0	7840.0	1.0	180.0	https://www.zillow.com/homedetails/1845-Lookin...
9	429900	3	3	2365	2	2.0	5227.0	1.0	240.0	https://www.zillow.com/homedetails/2109-Glen-H...
10	428000	4	3	2406	2	2.0	6969.0	1.0	176.0	https://www.zillow.com/homedetails/8045-Canonb...
11	625000	4	3	2941	3	2.0	12632.4	1.0	187.0	https://www.zillow.com/homedetails/109-Corbin-...

LASSO is a linear regression method that tries to minimize the the following equation:

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where y is the true value, the second sigma expresses the prediction, Beta is the coefficients, and lambda controls the amount of shrinkage. The important thing about LASSO is the 3rd sigma expression above which is the L1 Regularization. As the lambda value increases, ideally, it should sparsify the weights by pushing less important ones to 0.

The coordinate descent (CD) algorithm was used to minimize the cost function. The central idea of CD is that the cost function can be optimized by optimizing one direction at a time. The CD algorithm will first pick a direction. I used cyclic selection so the CD will iterate through all directions (or features). We'll call the coordinate it selected, j. It will then find a new Beta_j by minimizing the cost function with respect to j. All the other coordinate values are kept the same. This whole process will repeat until either maximum iterations is reached or it meets a convergence criterion (such as when the parameters don't change much anymore).

When I applied LASSO to my data, my mean absolute error from my cross validation score was 192353.907 dollars with a standard deviation of 169482.526 dollars. This poor score is probably due to my low number of samples. I also used data around my town, and if I was more specific about the neighborhood, it might have been better as well. Looking at my model coefficients:

Number of Bedrooms: 4.04951368e+04
Number of Bathrooms: 4.78772120e+04
square footage of house: 7.87661509e+01
numbers of cars in garage: 3.62891471e+04
number of stories: -3.92937578e+04
square footage of the lot: 1.15292990e+00
how many fireplaces: 3.69453519e+04
product of the kitchen dimensions: -1.46940668e+02

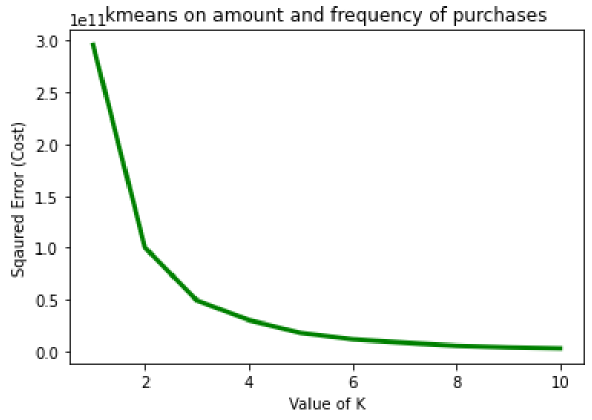
The weights are a little interesting since it negatively weighted the product of kitchen dimensions negatively, which I thought would be positive since a large kitchen usually means a more expensive home, and the number of stories having a large magnitude (since almost every house had 2 stories, I thought that would have become 0). Regardless, the number of bedrooms, bathrooms, parking garage space, number of stories, and how many fireplaces were the most important features to predict price according to my model.

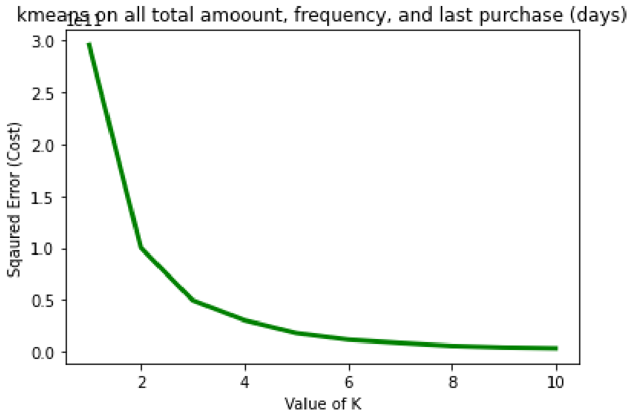
Problem 4

In this problem, we were given a kaggle dataset of all the transactions for an online store from 01/12/2010 to 09/12/2011. We were tasked with finding the best set of customers to target. Originally, the data is categorized by stock code number. Each row corresponds to a specific purchase made. The quantitative variables given were the quantity of the item purchased and the unit price of that item. Another feature was the date of the invoice.

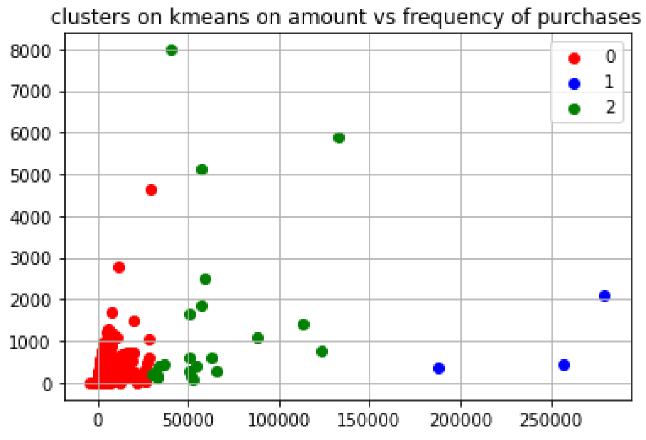
In order to effectively use k-means, I decided to do some feature engineering as it seems to be a standard approach for these types of problems. First, I dropped all rows with NaN values. Then, I created 3 features: the total amount (unit price * quantity) per customer, the total purchases made by customer, and the last purchase in terms of days ago from 0 (current) per customer. So, now I had a table of these 3 features where each row was a customer.

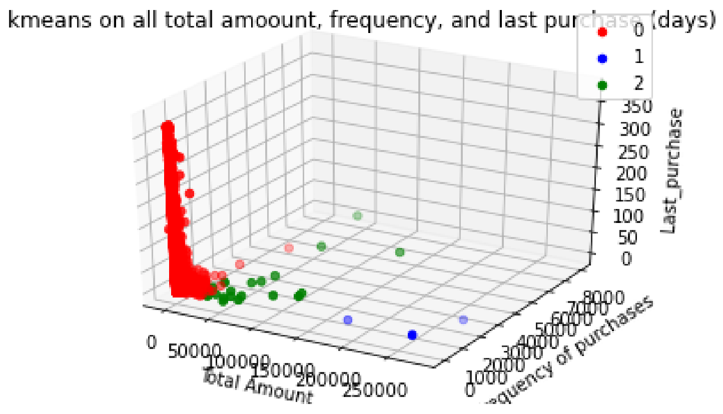
In order to find the optimal number of K to use, I created two graphs comparing the number of k vs the inertia (or the sum of the squared distances of samples to closest cluster center) and found the elbow points of these graphs. One graph was fitted on just the total amount and purchase frequency, while the other also included the time of the last purchase made.





The elbow point for both graphs was 3. I then plotted the data colored by their cluster.



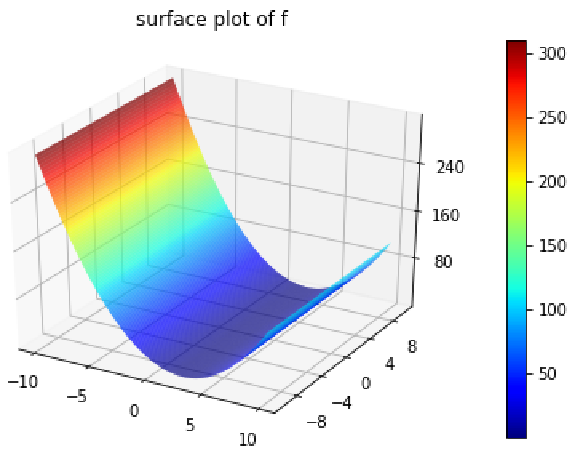


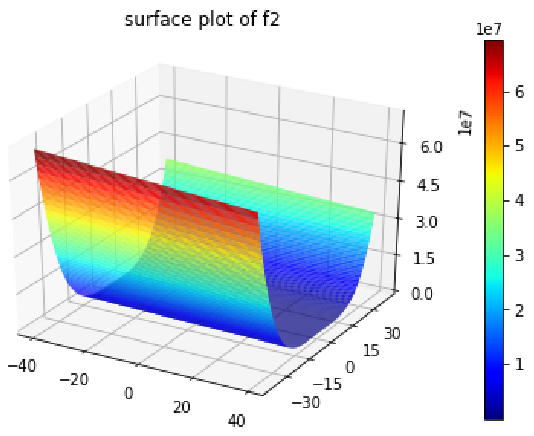
From the charts above, we can see that most of the customers (the red cluster) didn’t spend much, or buy often (for a wholesaler). Except for a few outliers in cluster 0, the customers the business should target are the customers in cluster 1 (blue) and cluster 2 (green). The customers in clusters 1 and 2 seem to make relatively few purchases, but they buy a large amount per invoice. They also tend to be regular customers since all the customers in cluster 1 and 2 seem to have bought within the (relative) past 100 days. Or, another way to look at it is that those are the regular customers and are happy with what the business offers. Thus, in order to increase customer retention, and maybe attract new customers, the business should look into the customers in cluster 0 and figure out why they aren’t purchasing more or why they haven’t been buying recently.

Problem 5

In this problem, we applied the gradient descent (GD) algorithm to two functions, f(x, y) = $(x - 2)^2 + (y - 3)^2$ and f2(x, y) = $[1 - (y - 3)]^2 + 20[(x + 3) - (y - 3)^2]^2$ and applied another variant of gradient descent, Nesterov's Accelerated Gradient descent (NAG) in my case, to the functions again.

The surface plots of the f and f2 are shown below:

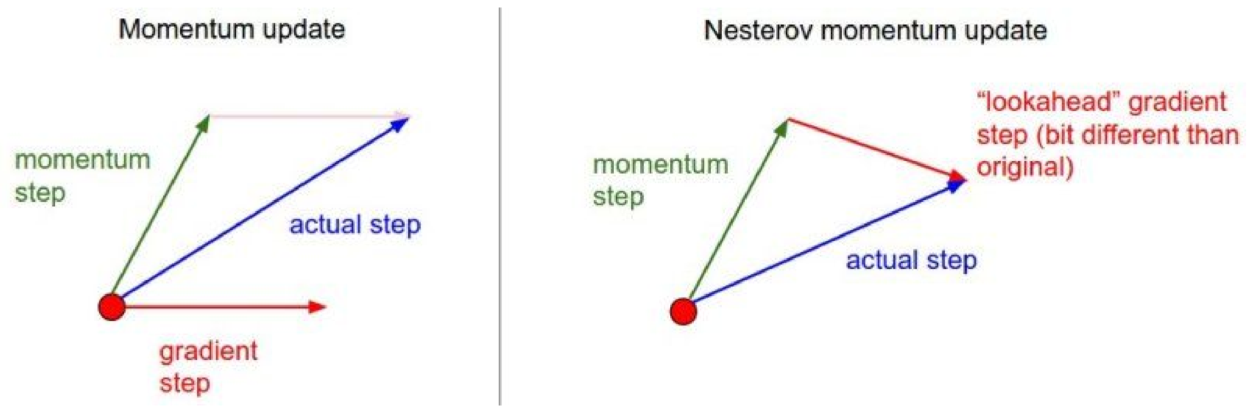




From the surface plots we can see that f2 is very steep compared to f. And this is apparent when we apply gradient descent with learning rate 0.5, starting values of (0,0) and 10 and 100 iterations for f and f2, respectively. For f, GD found the minimum point to be f(2,3) = 1. For f2, GD did not converge as x and y got too big for python. However, when the learning rate was lowered to 0.0001, GD found the minimum to be f2(0.52268091, 1.10840782) = 8.422777358620449.

NAG is a modification of gradient descent with momentum. In GD w/momentum, there is a velocity term that is parameterized by the momentum value and is what changes the (x,y). It's inspired by physics, where the momentum term acts more like a friction term to "slow down" a ball rolling down a hill. In other words, it dampens the learning rate * gradient value.

NAG modifies GD w/momentum by "looking ahead". Since we know how the momentum will change the previous velocity instead of calculating the gradient at the previous value, we calculate the gradient at the value that momentum changes the velocity by. The diagram below illustrates geometrically how it works:



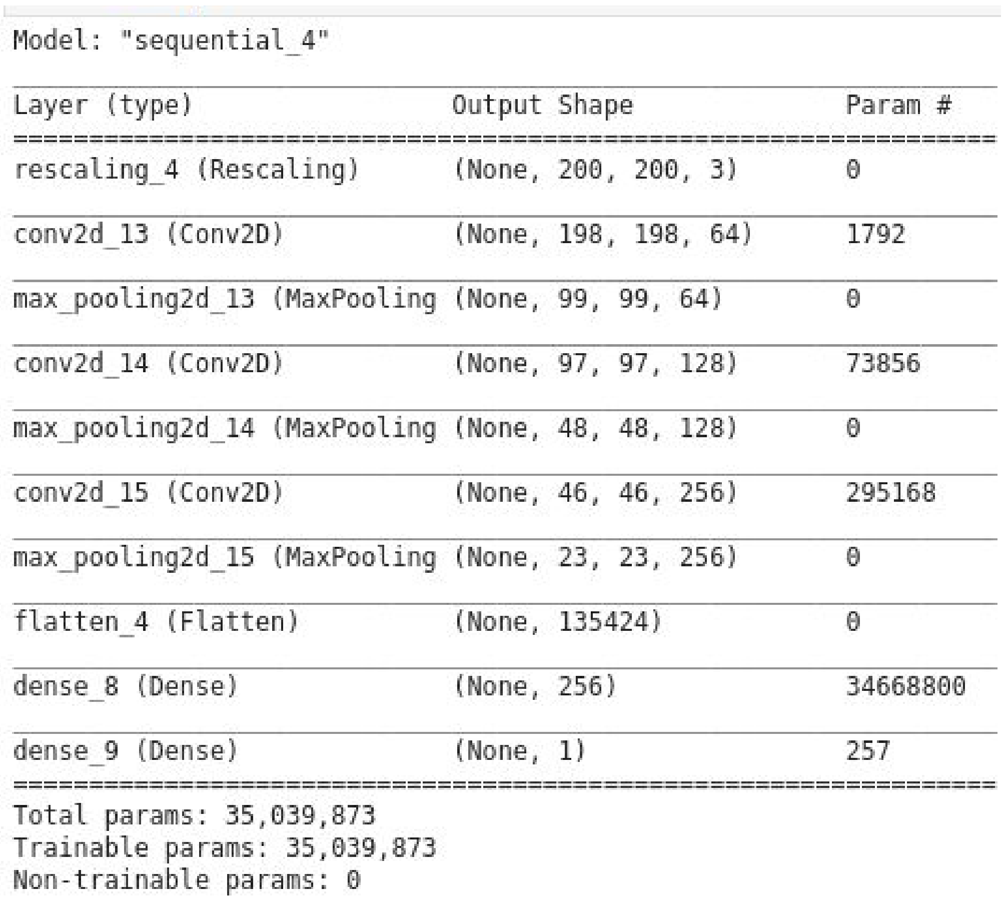
From "https://cs231n.github.io/neural-networks-3/#sgd"

I chose NAG because of the steepness of f2. I thought picking an algorithm that modifies the learning rate depending on the position on the loss surface would be more effective. And when applying NAG to our functions, it reached the same minimum as GD for f w/ learning rate at 0.5, but for f2, it found the minimum to be f2(-2.31541106, 2.190372) = 3.2916797367348396 at learning rate 0.001. So, NAG performed better than GD with a higher learning rate.

Problem 6

In this problem, we trained a CNN to classify whether a picture of a person was wearing a mask or not from a dataset on kaggle. Before we can apply the model though, we have to do some preprocessing. Since the images are all different sizes, they first had to be resized. I chose to resize them to 200 x 200. I also split the dataset into a train/test with the test being 20% of the data. The following images shows a sample of the resized image labeled with their class:

The CNN model architecture is:



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_4 (Rescaling)      (None, 200, 200, 3)       0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 198, 198, 64)      1792
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 99, 99, 64)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 97, 97, 128)       73856
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 48, 48, 128)       0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 46, 46, 256)       295168
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 23, 23, 256)       0
_________________________________________________________________
flatten_4 (Flatten)          (None, 135424)            0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               34668800
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 257
=================================================================
Total params: 35,039,873
Trainable params: 35,039,873
Non-trainable params: 0
```

Since it was a binary classification, the loss was Binary Crossentropy, and I used the adam optimizer. I ran for 5 epochs, and it achieved a training accuracy of 85.27% and a test accuracy of 85.43%. All the convolution filters were 3x3 with a stride of 1, and the max pooling filters were 2x2. I increased the number of filters for every convolutional layer since the earlier filters will learn smaller features from the pixel data, and later in the model, more features should help the network learn more abstractions.

I also applied the model to predict a picture of myself with and without a mask:

It predicted both successfully with the masked photo getting 0.08124608 (0 is with_mask) and the unmasked photo getting 0.8364424 (1 is without_mask).