**Predicting Mobile Application Success**

Team Members

Ola Mohsen Mohamed

Aya Abdul-Samie Abdul-Fattah

Mahmoud Gamal Abdul-Latif

Muhammad Medhat Fouad

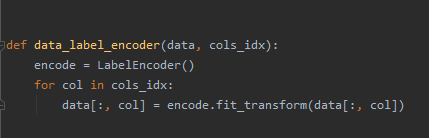
***Preprocessing Techniques***

* **Category Column:**

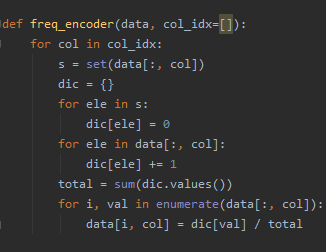
This column contains about 52 distinct values (including noise data). In categorical form ex {GAME\_SIMULATION, GAME\_CASUAL, SOCIAL}. There is no missing values.

Technique followed is to convert this unrelated categorical data to numeric form using label encoder

**Code snippets:**



Given numpy array of data and column indexes apply label encoder on all of these columns



Given numpy array of data and column indexes apply frequency encoder on all of these columns

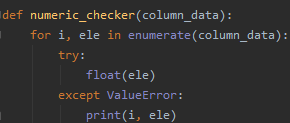
**Reviews Column:**

This Column contains 12672 distinct values. In numeric form ex {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0}.there is no noise in this column, but it contains nans

After applying a “numeric\_checker” function it does not report anything which means Reviews Column does not contain noise data, so it’s straightforward for impute missing values

No specific technique is followed in this column it’s ready after trimming, removing noisy data, impute nans if any exists and compensate nans by mean

**Code Snippets:**

Reports back if there is any value that’s cannot be transformed to numeric value

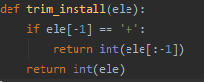
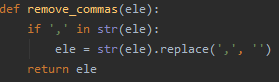
**Installs Column:**

This Column contains 24 distinct values. In categorical form ex { '10,000,000+', '0+', '1,000,000,000+', '11976'}. It does not contain any nans.

After applying “remove\_commas” and “trim\_installs” it reports that there is noise specifically in observation number ‘6941’ that has value of ‘EDUCATIONAL’

No specific technique is followed in this column it’s ready after trimming, removing noisy data and impute nans if any exists and compensate of nans by mean

**Code Snippets:**



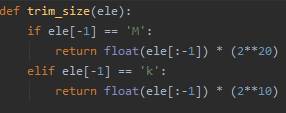
**Size Column:**

This Column contains 530 distinct values. In categorical form ex {'630k', '2.1M', '39k', '199k', '613k'}.it does not contain any nans.it contains a lot of missing values in form of ‘Varies with device’ so “trim\_min\_ver” is applied to compensate any such value with np.nan

After applying “remove\_commas” , “trim\_size” and “numeric\_checker” it turns out there is a noise data in this column specifically in observation 18477 that has value ‘1,000,000+’

No specific technique is followed in this column it’s ready after trimming, removing noisy data, impute nans if any exists and compensate of nans by mean

**Code Snippets:**



**Price Column:**

This Column contains 96 distinct values. In categorical form ex {'$3.49 ', '$10.99 ', '$31.99 ', '$10.00 '}. It does not contain any nans or missing values

After applying “trim\_price” it does not report for any noise values so it’s clean.

It’s just a matter to convert those strings to numeric values.

No specific technique is followed in this column it’s ready after trimming, removing noisy data, impute nans if any exists and compensate nans by mean

**Code Snippets:**



**Content Rate Column:**

This Column contains only 9 distinct values.so it’s easy to look at its values directly

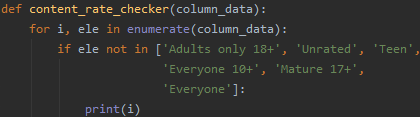


It’s obvious that there are some values that are not in the uniform.

After applying “content\_rate\_checker” to column data it turns out that the noisy data are in the following observations ‘6941’, ‘12624’ and ‘18477’ .

After removing noisy data and impute nans if any exists with most frequent strategy, convert unrelated categorical data to numeric form using label encoder, then assign to each distinct value a rate base on how frequently it’s appear in column using “data\_label\_encoder”

**Code Snippets:**

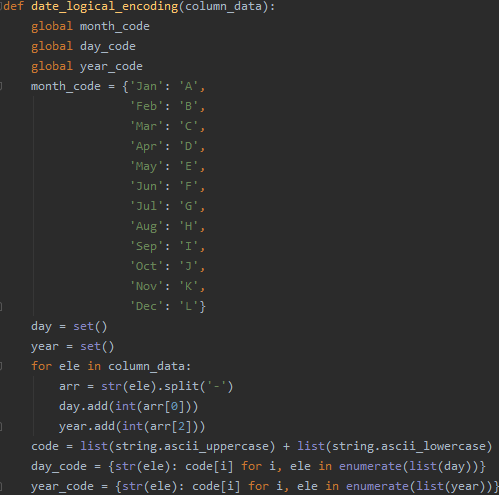


**Last Updated Column:**

This Column contains 1728 distinct values. There are no noisy data or nans. It’s data in categorical form ex {'1-Jul-16', '12-Feb-15', '16-Aug-16'}. It’s clean column, however this column had a very special care while dealing with it since it’s categorical data that holds relationship among themselves and this relationship must hold after encoding them to numeric values, if we have assumed a weight assigned to each date let’s say that’s the newer the date the higher the weight (capture the intuition that people likes updated things so it’s approved thing).

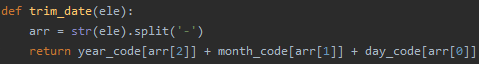
Label encoder in python does not care about relational categorical data it’s just put them in numeric values based their lexicographical order, but this does not hold the relation for example it might assign to 5-Dec-18 a higher value than 1-Jan-19 but this is so good so we had to encode them to values that holds this relation and here where “data\_logical\_encoder” comes

**Code Snipptes :**



it encodes the dates in a way that preserve its chronological order namely higher weight to closer ones

then comes trimming those dates based on their code using “trim\_date” function



After that Label encoder then encodes this information in the desired way.

**Minimum Version Column:**

This column contains 34 distinct values.so it’s easy to look at it’s values directly and it turns out it’s categorical unrelated data and there are no noisy data, however there are a lot of missing values in the form of ‘Varies with device’

So it’s compensated using imputer with most frequent strategy and then encoded using label encoder

**Latest Version Column:**

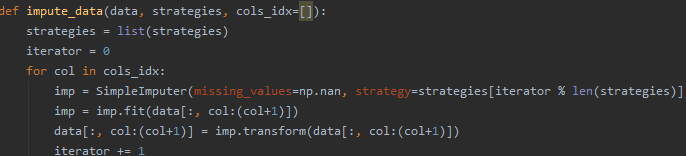
This Column contains 5768 distinct values. it’s a categorical unrelated data that contains nans and ‘Varies with device’ words , we decide to drop it since it will not add any useful information to our model .

**Dealing with noisy data** after taking the unions of all noisy data (data is not in the uniform) we found only three noisy observations (‘6941’, ‘12624’ and ‘18477’) so a decision is taken to drop them

**Code Snippets:**

****

Imputer function

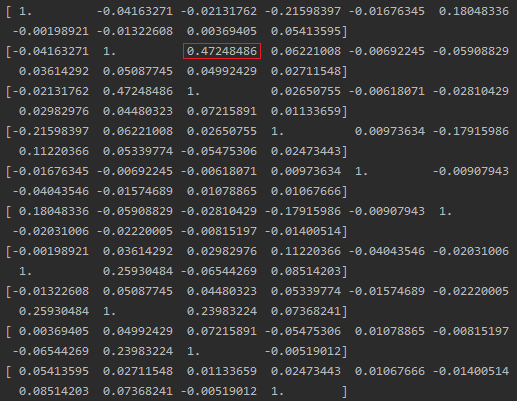


**App Name Column:**

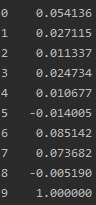
This column contains higher than 20,000 distinct values so it’s obvious will not add any useful information to our model so it’s discarded.

**Dataset Analysis**

From analyzing data frame correlations



It turns out the data columns correlate very poorly to each other except a correlation between Reviews and Installs column which seems to be relatively good compared to the others



But looking at the very important values to us that’s the correlations between dependent variable and the independent ones it’s obvious that they are poorly correlated to the dependent variable

**Regression Techniques**

**Multi-Variant Linear Regression**

In such a model if we have n features then we are trying to fit n-1 dimensional object to our data that minimizes the cost function

H(x) = theta0 + theta1 \* X1 + theta2 \* X2 +…+ thetan \* Xn-1

J(theta) = (1/2m) \* sum((y\_predicted – y\_actual)\*\*2)

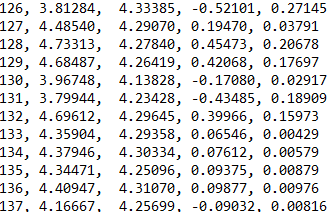
In context of our model, all features except “app name” will go to an automatic feature selection algorithms based on correlation between independent variables and the dependent variable which will select features to use based on a specific threshold which is typically 0.006

namely, any feature that has a correlation that is less that that threshold will be ignored

**results :**

SSE = 1304.163

MSE = 0.252



a small sample of multi-variant model (index, y\_actual, y\_predict, error, square error)

**Polynomial Linear Regression**

In such a model we are trying to fit a higher order polynomial to our data

In order to make a better prediction we are trying to minimize the cost function

J(theta) = (1/2m) \* sum((y\_predicted – y\_actual)\*\*2)

Under the light of hypothesis function

H(x) = theta0 + theta1 \* X1 + theta2 \* X2 + theta3 \* X1 \* X2 + theta4 \* X12 + theta5 \* X22 + …..

And so on so forth with higher order

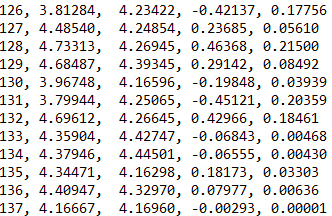
In context of our model, all features except “app name” will go to a feature generator to introduce new features of our desired polynomial degree then automatic feature selection algorithm based on correlation between independent variables and the dependent variable which will select features to use based on a specific threshold which is typically 0.006

namely, any feature that has a correlation that is less that that threshold will be ignored

**results :**

SSE = 1274.394

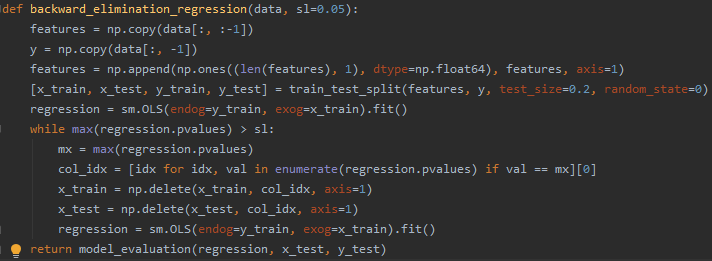
MSE = 0.246

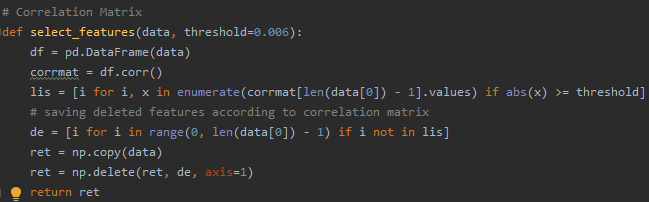


a small sample of polynomial 3 model (index, y\_actual, y\_predict, error, square error)

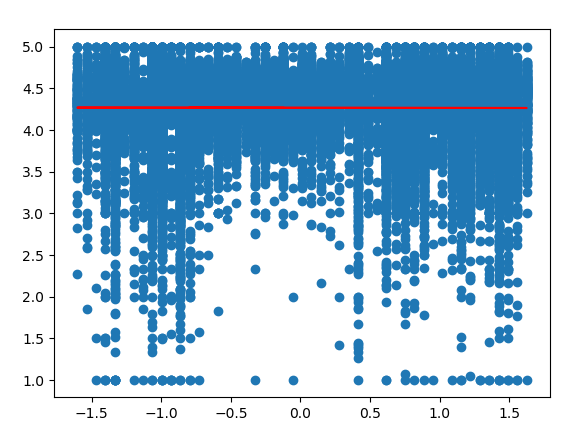
we relied on an automatic feature selection basically two algorithms correlation determination and backward elimination

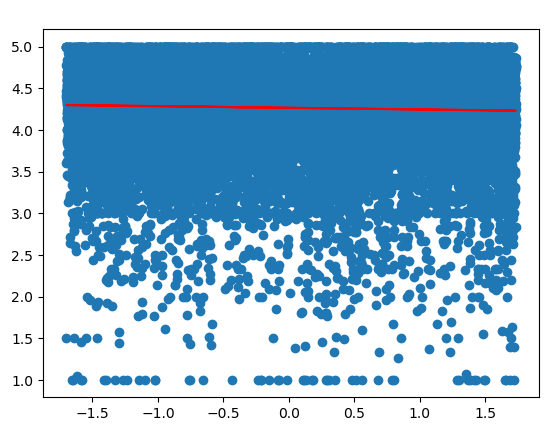
**Code Snippets:**

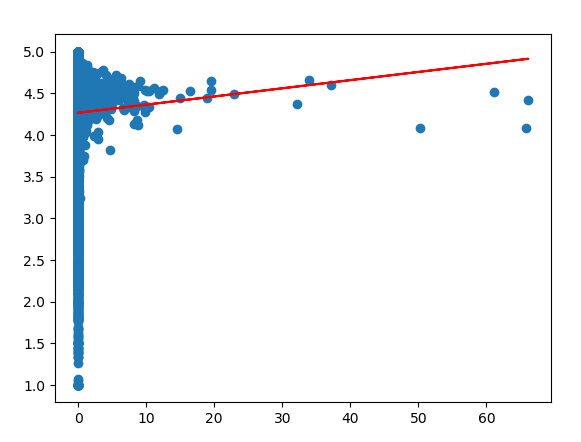
****

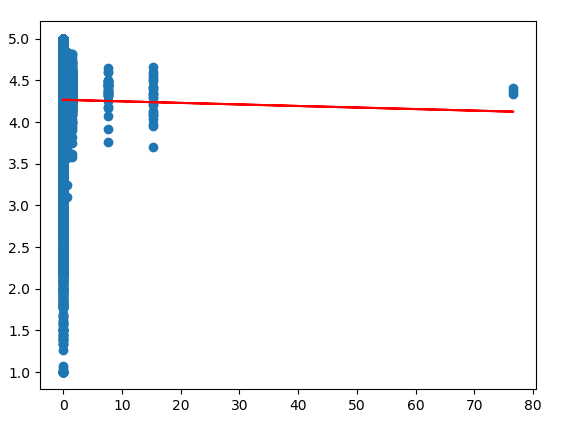


in purpose of our models the dataset is splitted into two parts training and testing with 80% and 20% respectively

Plotting: 







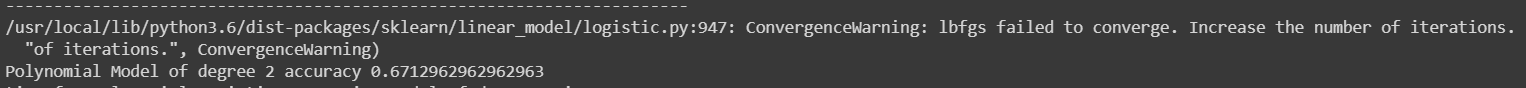
**Report Part 2**:

**Logistic Regression:**

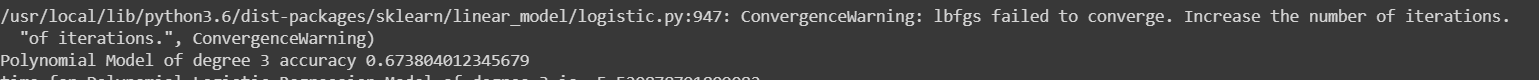
In **Polynomial logistic Regression** Model When using parameters **solver = ‘lbfgs’ , multi\_calss = ‘auto’**



It gives a warning that solver lbfgs failed to converge with accuracy in degree 2 : 0.6712962962962963



And for degree 3 it gives the same warning with accuracy : 0.673804012345679



Where when using **solver** = ‘***newton-cg’ , multi\_calss = ‘multinomial’***



It runs without any warnings and for degree 2 it gives accuracy : 0.6714891975308642 which is slightly increased by 0.0002



And also for degree 3 the accuracy becomes : 0.6763117283950617 which is higher than the one above with 0.003



So, we decided to set **solver** = ‘***newton-cg’ , multi\_calss = ‘multinomial’***  because this gives the highest accuracy since multinomial minimizes the loss fit across the entire probability distribution and ***newton-cg*** to handle multinomial loss

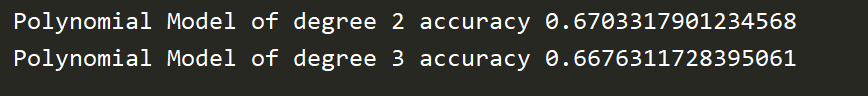
**Polynomial Logistic Regression After applying PCA:**

we have tried to select features from 1 to 10 what we have noticed is that when selecting 6 features we get the best accuracy but, it failed to converge in polynomial of degree 3;

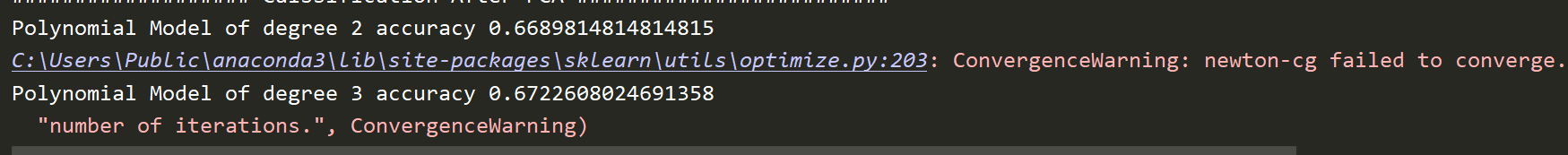
When we increase selected features it still fails to converge at degree 3 also the accuracy decreases too where when decrease the number of selected features the accuracy decreases although, it starts to converge at degree 3 when select 3 features or below .

Here are some of our test

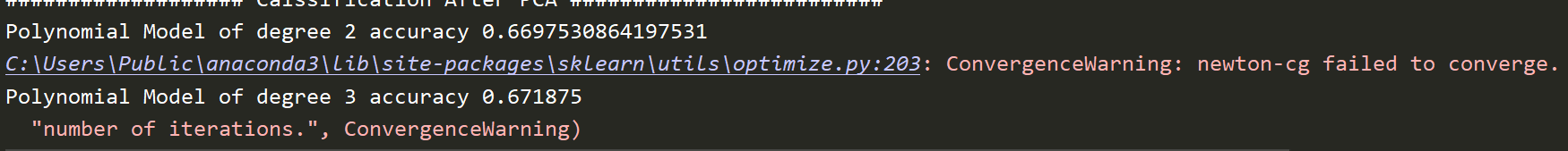
When selecting 3 features the accuracy is



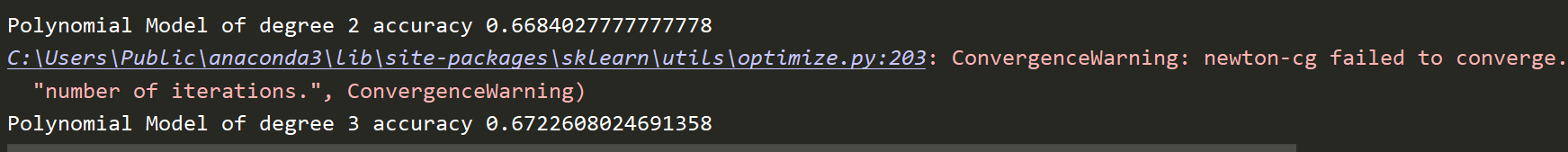
When selecting 6 features it the accuracy is



When selecting 7 features it gives accuracy



When selecting 8 features



From the above observation we noticed that when selecting 6 features this gives the best accuracy for degree 3 but, it fails to converge where when select 7 features gives higher accuracy for degree 2.

One of the solutions to solve the convergence problem is to increase the number of iterations but, when we tried this solution the code never stopped running that’s why we excluded this solution .

**Adaboost:**

In **Adaboost** without applying PCA using decision tree when setting hyper parameter algorithm = ‘SAMME’, n\_estimators = 200 the accuracy was 0.6772762345679012



And when applying PCA the accuracy decreased to be : 0.6649305555555556



When changing hyper parameter algorithm to be = ‘SAMME.R’ , n\_estimators = 200



the accuracy becomes : 0.6412037037037037 (without applying PCA)



Also when decreasing the value of n\_estimators to 100



The accuracy decreased to be : 0.6342592592592593



When applying PCA to Adaboost with hyper parameters be algorithm = ‘SAMME.R’ , n\_estimators = 200

the accuracy becomes : 0.6564429012345679



When decreasing n\_estimators to be 100



The accuracy becomes : 0.6651234567901234



When setting algorithm = ‘SAMME.R’ , n\_estimators = 50



Without PCA the accuracy becomes : 0.6844135802469136



When applying PCA it becomes : 0.6670524691358025



For algorithm = ‘SAMME’ , n\_estimators = 50



The accuracy before applying PCA was



And after applying PCA becomes



So, we finally decided to set algorithm = ‘SAMME.R’ , n\_estimators = 50 which is the best observation

Also because SAMME.R algorithm is faster to converge than SAMME algorithm, achieving a lower test error with fewer boosting iterations and When we start to decrease n\_estimator to be under 50 or above 50 the accuracy decreased too

**Here is our final code**



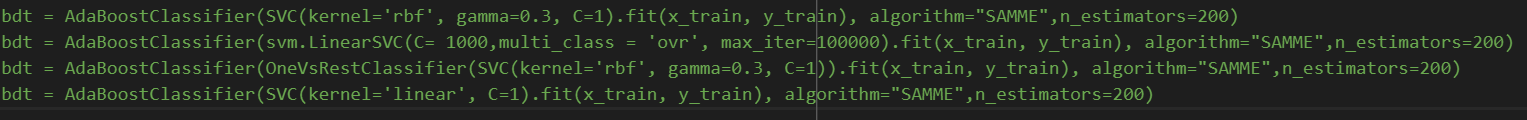
And our final accuracy before applying PCA



The accuracy after applying PCA

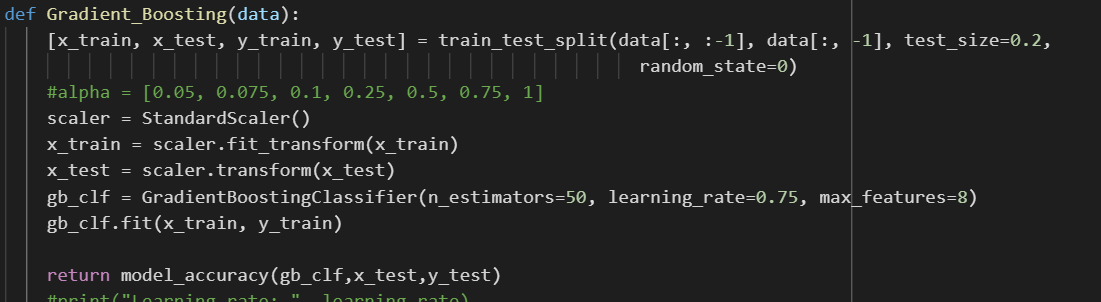


When we tried ADABOOST on SVM (One Vs One , One Vs Rest , Linear , without kernals) it did not work as it take too much time to run and it did not stop at the end which we did not find an explanation for this and based on this observation we decided to exclude it



For Logistic Regression with ADABOOST it also did not stop and when we search for the reason we found out a new algorithm that work as ADABOOST for logistic regression which is Gradiant Boosting it works typically as ADABOOST it combines weak "learners" into a single strong learner in an iterative fashion but, what differs is that ADABOOST works on classifiers that give output (0,1), where logistic regression outputs probabilities from (-1 to 1) that’s why we apply Gradiant boosting it minimizes the differentiable loss functions by constructing a new model that adds an estimator  to provide a better model.

 Gradiant Boosting shows how sklearn code allows for a choice between deviance loss for logistic regression and exponential loss for AdaBoost, and documents functions to predict probabilities from the gradient-boosted model.



We have tried random values (0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1) for the learning rate and what we have noticed is that 0.75 gives the best accuracy.

Gradiant Boosting have its own way for selecting features which is parameter max\_features we have tried from 1 to 10 for selecting features and what we have got is that selecting 8 features gives the best accuracy.



**Decision Tree:**

When setting max\_depth = 1



The accuracy was the same before and after applying PCA





And when setting max\_depth = 2



The accuracy increased by 0.003 before applying PCA and increased by 0.002





But when we increased max\_depth to be 3

The accuracy increased before applying PCA to be



But decreased by 0.0009 after applying PCA to be



When we increased max\_depth to be 4



The accuracy increased in both before and after applying PCA





We noticed that when increasing max\_depth to be above 4 the accuracy after applying PCA decreases and increases before applying PCA

So, we finally decided to set max\_depth to be 4

**KNN:**

## **Before PCA:**

When applying KNN with different value for ( number of neighbors) **k**=**1** to **200** , we have noticed that the best accuracy was given when **k**=**72** with accuracy = 0.6674382716049382‬

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generatedA close up of text on a white background

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**After PCA:**

The Best accuracy was the same before PCA with also **k**=72

**SVM**:

**Gaussian Kernel**:

**Before applying PCA**

When applying SVM via **One VS Rest (OVR)** method with a **Gaussian** Kernel and with hyperparameter **sigmma (σ)** =**0.2** with the same **c** the accuracy was equal 0.6660879629629



Also it was observed that when **σ** = **0.3 ,0.4, 0.5 , 0.6** or **0.7** with the same **c**  the accuracy increased by 0.00019 and was equal to 0.6662808641975309



and when **c**=1 and **σ**=0.4 the accuracy with also equal 0.6662808641975309



Then when we set **σ**=0.2 and change **c** with values : 100 ,1000,10000 ,the accuracy decreased and become 0.6660879629629629



After that we apply SVM via **One VS One (OVO)** method with **σ=**0.4 and **c**= 1, 100 and 1000000



It gives the same accuracy of OVR with **σ**=0.4 and **c**=1 , but since that the training data required for each class is balanced in OVO and we have only 3 classes , also The training and testing time was less than OVR , so we decide to apply SVM via OVO with **σ=0.4** and **c=1**

****

****

**After applying PCA:**

with **n\_components=7 ,**we observed that the accuracy obviously increased to become = 0.6745756172839507 inOVRand become **=** 0.6745756172839507 in OVO



and when **n\_components=9 it** decreased in OVR and become **=** 0.673804012345679 in OVR but increased in OVO and become =0.6751543209876543



when **n\_components=9 it** gives the best accuracy in OVR and become **=** 0.6739969135802469 and in = OVO become =0.6763117283950617



So ,the conclusion is that SVM gives its best accuracy when applying PCA with n\_components=9

**Linear Kernel**:

**Before PCA:**

It takes too long time to give the results

**After PCA :**

It gives an accuracy which is equal to 0.6662808641975309 so it does not increase more than SVM with Gaussian kernel , so we decide to not apply it



**Without Kernel:**

**Before PCA:**

It gives a bad accuracy which is equal to 0.4525462962962963



**After PCA:**

It takes a long time to run so we decide to not apply SVM without kernel especially since it gives bad accuracy before PCA

**Selective Feature:**

We used ‘**SelectKBest’** Selection Method and give it ‘**ANOVA**’ as statistical measure

**Polynomial Logistic Regression:**

when trying different values for no. of features (**K**) , we noticed that the accuracy didn’t get better , On the Country it was less than the accuracy before selective feature.



**Other Models**:

We tried to assign values for **K** (Number of selected feature) from 1 to 10 and we got that is the best accuracy was given when we select all the ten features.

The best two accuracy after k=10 was when k=9 and k=6.

When **k= 6**

A close up of a newspaper

Description automatically generated

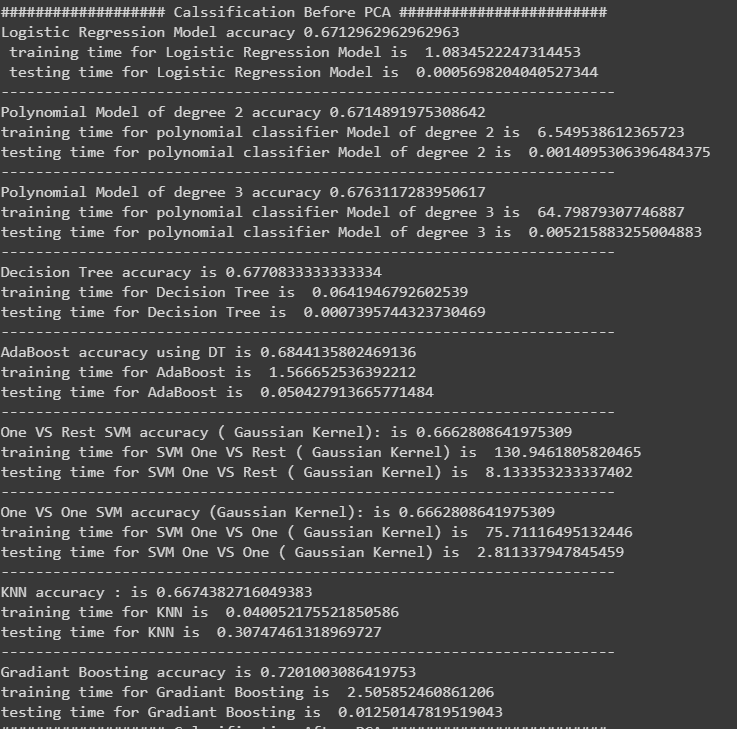
When **k=9**

A close up of a newspaper

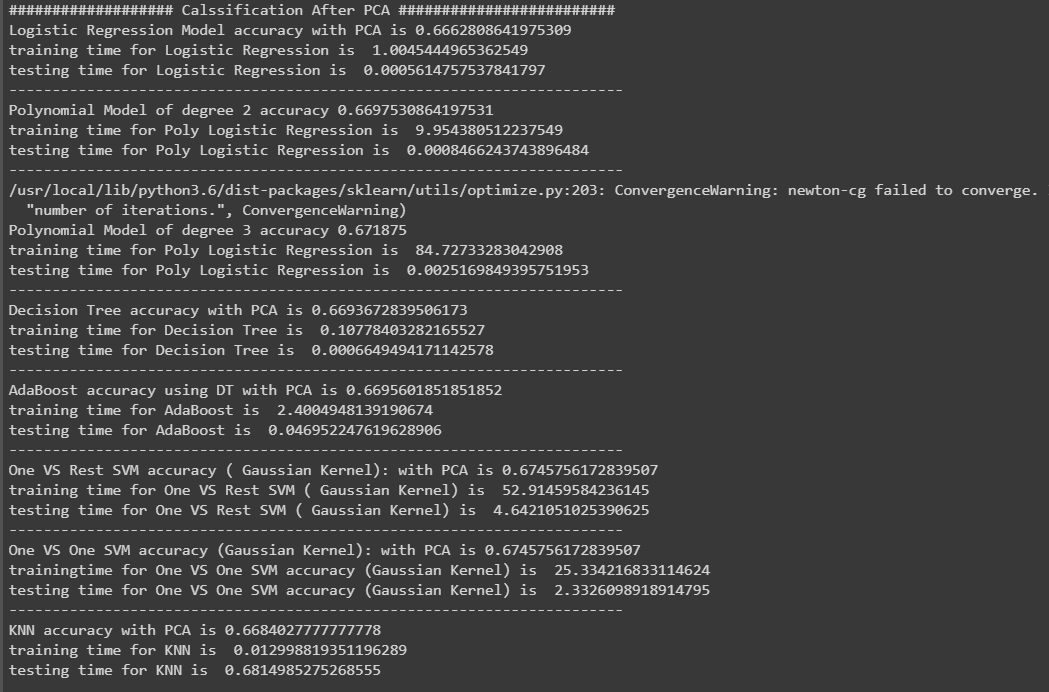
Description automatically generated

**PCA:**

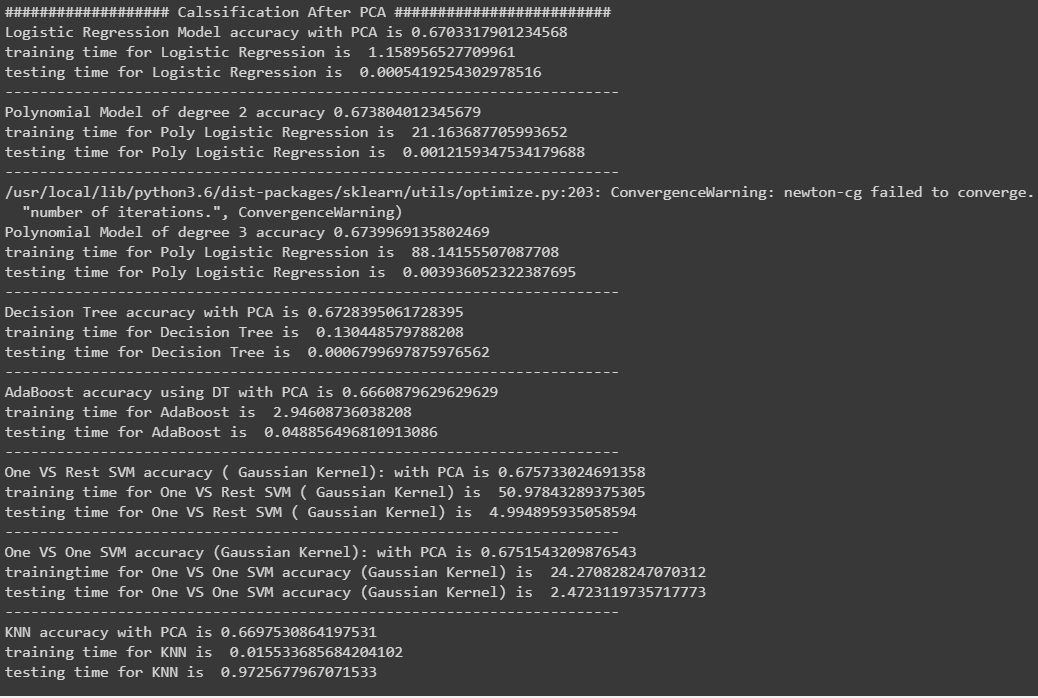
The accuracy before PCA



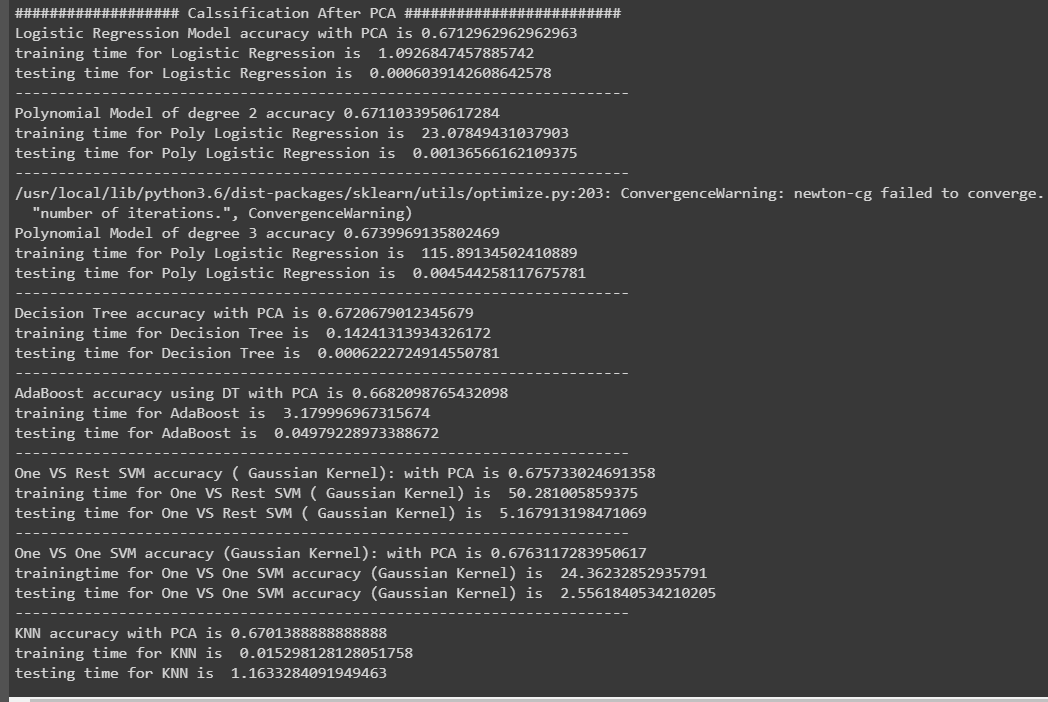
When n\_components = 7



When n\_estimsators = 9



When n\_estimators = 10



From trials we noticed that the best 3 values of selected features is 7 , 9 , 10 because they give the best accuracy since for the observation over the 3 values is as follows :

The 3 values increase the accuracy for SVM One Vs Rest , SVM One Vs One and KNN Models with some differences

1. For 7 features gives the highest accuracy for **AdaBoost** and **SVM One Vs Rest** classifiers
2. For 9 features gives the best accuracy for **Polynomial Regression of (degree 2 , degree 3)** and **ADABOOST**
3. For 10 features gives the best accuracy for **Logistic Regression** , **Polynomial logistic regression of degree 3, KNN** and **SVM One Vs One**

**Obviously, the observation shows that selecting 10 features is the best choice**.

**NOTE:** we decided to leave polynomial logistic regression of degree 3 with its warning because when we tried to solve the convergence problem by increasing number of iterations it did not stop running.

**Some Important Notes about Saving the Models:**

When saving the models ,and test with the loaded models we observed that the accuracy has changed Significantly . A slight change was expected but the change was large which is a little strange.

Another thing to be mentioned that when we changed the ‘***shuffle’*** parameter , and test with the loaded models , another large changes were occur , here is the details:

* When shuffling , splitting the data and saving the models , then loading them to test after shuffling and splitting the data again , that was the accuracy of each model:

A close up of text on a black background

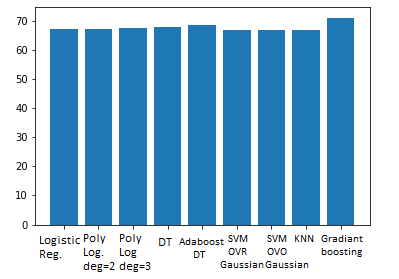
Description automatically generated

* When shuffling , splitting the data and saving the models ,then loading them to test after **only** splitting them again and not making shuffle , these were the results:
* A close up of text on a black background

  Description automatically generated

**Plots:**

* **Before applying PCA:**

**** *A screenshot of a cell phone

Description automatically generated*

Figure .Accuracy Figure .Training Time

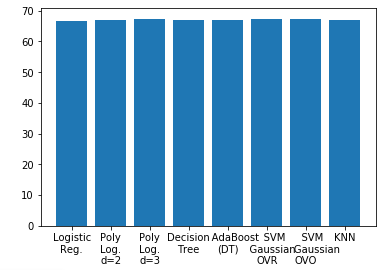
**A screenshot of a cell phone

Description automatically generated**

Figure .Testing Time

* **After applying PCA:**

When PCA=7

A screenshot of a cell phone

Description automatically generated

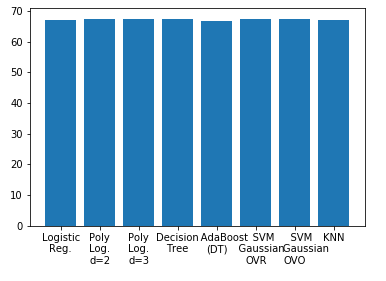
Figure .Accuracy Figure .Training Time

A screenshot of a cell phone

Description automatically generated

Figure .Testing Time

When PCA=9

A screenshot of a cell phone

Description automatically generated

Figure 7.Accuracy Figure .Training Time

A screenshot of a cell phone

Description automatically generated

Figure 9.Testing Time

When PCA=10

A screenshot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

Figure 10.Accuracy Figure .Training Time

A screenshot of a cell phone

Description automatically generated

Figure 12.Testing Time