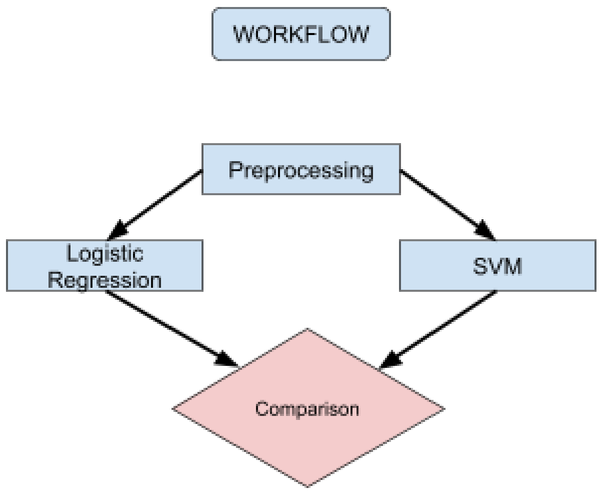
IE 585 PROJECT REPORT

Group 4:
İlayda Çelenk - 2019702147
Arifcan Yiğit - 2020802036
Other members dropped the course.

Contents:

1. Introduction and Preprocessing



Preprocessing part includes the work using the provided script called 1.preprocessing.R.
For the Logistic Regression method, 2.logistic_regression.R script is provided.
For the SVM method 3.svm.ipynb is provided.
Comparison is done in the report using the outcomes from the scripts. In the preprocessing part, datasets are read and analyses are made.

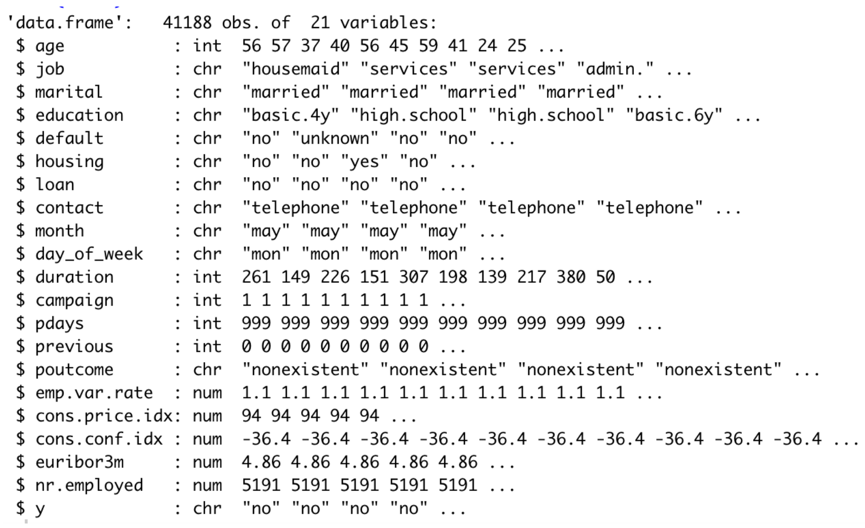
First Dataset:

AI4I 2020 Predictive Maintenance Data Set consists of information related to maintenance encountered in industry. The aim is to predict if there is a machine failure or not. The first 2 columns (UDI and Product.ID) are unique therefore they are dropped. The last 5 columns are the types of failures, and they are dropped since if any of them is 1 then the failure will be 1. Type column is nominal and there are 3 types, therefore one-hot-encoding is used to create 3 columns instead of Type. Then one column is dropped since it is deterministic by the other 2. For the numerical features, scaling is done.
So finally 7 features are used to determine the target(Machine.failure) which is transformed to {-1, 1}. Here -1 indicates no failure and 1 indicates failure. There are 10000 observations. The dataset is imbalanced since failures do not happen frequently. Using *skim* function from *skimr* library, statistics can be observed and there are no missing values.

```
'data.frame':	10000 obs. of  14 variables:
 $ UDI                    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Product.ID             : chr  "M14860" "L47181" "L47182" "L47183" ...
 $ Type                   : chr  "M" "L" "L" "L" ...
 $ Air.temperature..K.    : num  298 298 298 298 298 ...
 $ Process.temperature..K.: num  309 309 308 309 309 ...
 $ Rotational.speed..rpm. : int  1551 1408 1498 1433 1408 1425 1558 1527 1667 1741 ...
 $ Torque..Nm.            : num  42.8 46.3 49.4 39.5 40 41.9 42.4 40.2 28.6 28 ...
 $ Tool.wear..min.        : int  0 3 5 7 9 11 14 16 18 21 ...
 $ Machine.failure        : int  0 0 0 0 0 0 0 0 0 0 ...
 $ TWF                    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ HDF                    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ PWF                    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ OSF                    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ RNF                    : int  0 0 0 0 0 0 0 0 0 0 ...
```

Second Dataset:

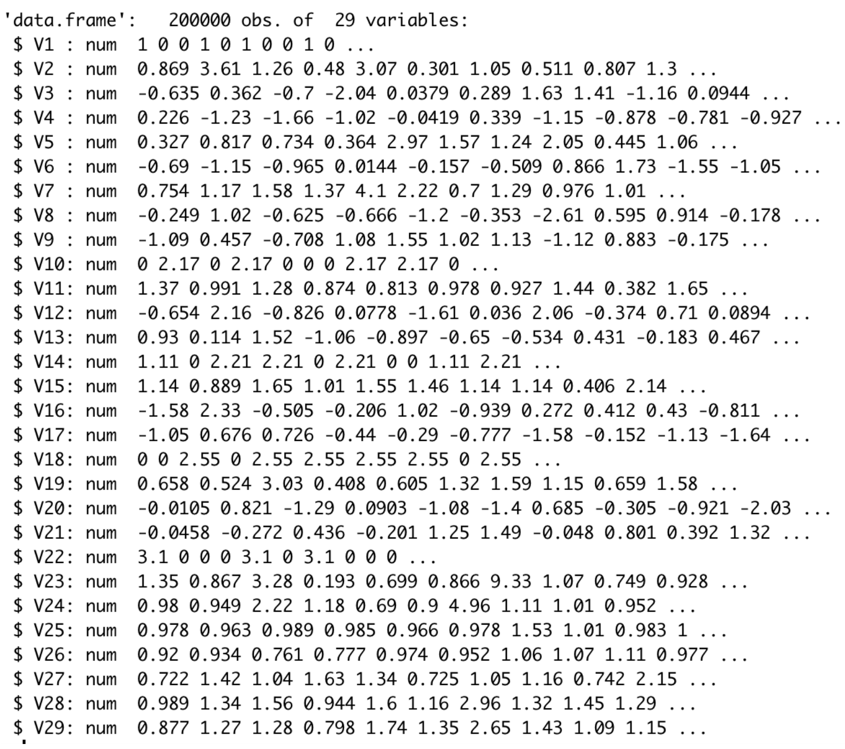
Bank Marketing Data Set consists of information that reflects direct marketing campaigns of a bank. The aim is to predict if bank term deposit is subscribed or not.
There are 10 categorical features with various unique values, one-hot-encoding with dropping one column technique is used for each of them since they are nominal. For the numerical features, scaling is done.
So finally 53 features are used to determine the target(y) which is transformed to {-1, 1}. Here -1 indicates no subscription and 1 indicates subscription. There are 41188 observations.
Some categorical features may consist of correlated values. They could be grouped in order to decrease the number of columns created at one-hot-encoding processes. Using *skim* function from *skimr* library, statistics can be observed and there are no missing values.



```
'data.frame':	41188 obs. of  21 variables:
 $ age           : int  56 57 37 40 56 45 59 41 24 25 ...
 $ job           : chr  "housemaid" "services" "services" "admin." ...
 $ marital       : chr  "married" "married" "married" "married" ...
 $ education     : chr  "basic.4y" "high.school" "high.school" "basic.6y" ...
 $ default       : chr  "no" "unknown" "no" "no" ...
 $ housing       : chr  "no" "no" "yes" "no" ...
 $ loan          : chr  "no" "no" "no" "no" ...
 $ contact       : chr  "telephone" "telephone" "telephone" "telephone" ...
 $ month         : chr  "may" "may" "may" "may" ...
 $ day_of_week   : chr  "mon" "mon" "mon" "mon" ...
 $ duration      : int  261 149 226 151 307 198 139 217 380 50 ...
 $ campaign      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ pdays         : int  999 999 999 999 999 999 999 999 999 999 ...
 $ previous      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ poutcome      : chr  "nonexistent" "nonexistent" "nonexistent" "nonexistent" ...
 $ emp.var.rate  : num  1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 ...
 $ cons.price.idx: num  94 94 94 94 94 ...
 $ cons.conf.idx : num  -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 ...
 $ euribor3m     : num  4.86 4.86 4.86 4.86 4.86 ...
 $ nr.employed   : num  5191 5191 5191 5191 5191 ...
 $ y             : chr  "no" "no" "no" "no" ...
```

Third Dataset:

HIGGS Data Set consists of 28 numerical columns, where the first 21 features are kinematic properties measured by the particle detectors in the accelerator and the last 7 are derived from them by physicists.
Since all the features are numerical, scaling is done for all of them.
So finally 28 features are used to determine the target(V1) which is transformed to {-1, 1}. Here -1 indicates background and 1 indicates signal. There are 200000 observations in the data set file provided. Using *skim* function from *skimr* library, statistics can be observed and there are no missing values.



```
'data.frame':	200000 obs. of  29 variables:
 $ V1 : num  1 0 0 1 0 1 0 0 1 0 ...
 $ V2 : num  0.869 3.61 1.26 0.48 3.07 0.301 1.05 0.511 0.807 1.3 ...
 $ V3 : num  -0.635 0.362 -0.7 -2.04 0.0379 0.289 1.63 1.41 -1.16 0.0944 ...
 $ V4 : num  0.226 -1.23 -1.66 -1.02 -0.0419 0.339 -1.15 -0.878 -0.781 -0.927 ...
 $ V5 : num  0.327 0.817 0.734 0.364 2.97 1.57 1.24 2.05 0.445 1.06 ...
 $ V6 : num  -0.69 -1.15 -0.965 0.0144 -0.157 -0.509 0.866 1.73 -1.55 -1.05 ...
 $ V7 : num  0.754 1.17 1.58 1.37 4.1 2.22 0.7 1.29 0.976 1.01 ...
 $ V8 : num  -0.249 1.02 -0.625 -0.666 -1.2 -0.353 -2.61 0.595 0.914 -0.178 ...
 $ V9 : num  -1.09 0.457 -0.708 1.08 1.55 1.02 1.13 -1.12 0.883 -0.175 ...
 $ V10: num  0 2.17 0 2.17 0 0 0 2.17 2.17 0 ...
 $ V11: num  1.37 0.991 1.28 0.874 0.813 0.978 0.927 1.44 0.382 1.65 ...
 $ V12: num  -0.654 2.16 -0.826 0.0778 -1.61 0.036 2.06 -0.374 0.71 0.0894 ...
 $ V13: num  0.93 0.114 1.52 -1.06 -0.897 -0.65 -0.534 0.431 -0.183 0.467 ...
 $ V14: num  1.11 0 2.21 2.21 0 2.21 0 0 1.11 2.21 ...
 $ V15: num  1.14 0.889 1.65 1.01 1.55 1.46 1.14 1.14 0.406 2.14 ...
 $ V16: num  -1.58 2.33 -0.505 -0.206 1.02 -0.939 0.272 0.412 0.43 -0.811 ...
 $ V17: num  -1.05 0.676 0.726 -0.44 -0.29 -0.777 -1.58 -0.152 -1.13 -1.64 ...
 $ V18: num  0 0 2.55 0 2.55 2.55 2.55 2.55 0 2.55 ...
 $ V19: num  0.658 0.524 3.03 0.408 0.605 1.32 1.59 1.15 0.659 1.58 ...
 $ V20: num  -0.0105 0.821 -1.29 0.0903 -1.08 -1.4 0.685 -0.305 -0.921 -2.03 ...
 $ V21: num  -0.0458 -0.272 0.436 -0.201 1.25 1.49 -0.048 0.801 0.392 1.32 ...
 $ V22: num  3.1 0 0 0 3.1 0 3.1 0 0 0 ...
 $ V23: num  1.35 0.867 3.28 0.193 0.699 0.866 9.33 1.07 0.749 0.928 ...
 $ V24: num  0.98 0.949 2.22 1.18 0.69 0.9 4.96 1.11 1.01 0.952 ...
 $ V25: num  0.978 0.963 0.989 0.985 0.966 0.978 1.53 1.01 0.983 1 ...
 $ V26: num  0.92 0.934 0.761 0.777 0.974 0.952 1.06 1.07 1.11 0.977 ...
 $ V27: num  0.722 1.42 1.04 1.63 1.34 0.725 1.05 1.16 0.742 2.15 ...
 $ V28: num  0.989 1.34 1.56 0.944 1.6 1.16 2.96 1.32 1.45 1.29 ...
 $ V29: num  0.877 1.27 1.28 0.798 1.74 1.35 2.65 1.43 1.09 1.15 ...
```

Each dataset is divided into train, validation and test sets. For the train sets 60% of the observations are used, for the validation sets 20% of the observations are used and for the test sets 20% of the observations are used.

So finally there are 9 datasets created in this R script and they are exported as .csv files. Instead of fixing the seed, these files will be read in the scripts of the predictive models so that all the models take the same train-validation-test observations.

2. Logistic Regression with L2 regularization using stochastic gradient algorithms

Firstly, logistic regression was used to classify the data points while finding the MLE of its parameters using the basic stochastic gradient algorithm (SGD). Although SGD converges in expectation, the duration may last long. The reason is that parameters are adjusted with the new information from all points one by one, which may in turn give opposite, irrelevant directions in case of outliers or due to pure randomness of the data. After decreasing loss below a level, SGD can continue its adjustments to reflect all data points.

For the mentioned reasons, the momentum and averaging variants of SGD were also considered. Momentum variant aims to reflect a certain percentage of previous update also in the current update.[1,2] By this way, zig zag behaviour until convergence is reduced. The eta is an additional parameter and referred to as momentum or momentum coefficient in the code and the report.

$$\Delta w := \alpha \Delta w - \eta \nabla Q_i(w)$$
$$w := w + \Delta w$$

The averaging variant SAG keeps a record of gradients evaluated at all points. Then updates the gradient of the considered point by the current information and uses the average of all gradients to compute new parameter estimation. [3]

$$\bar{w} = \frac{1}{t} \sum_{i=0}^{t-1} w_i \,.$$

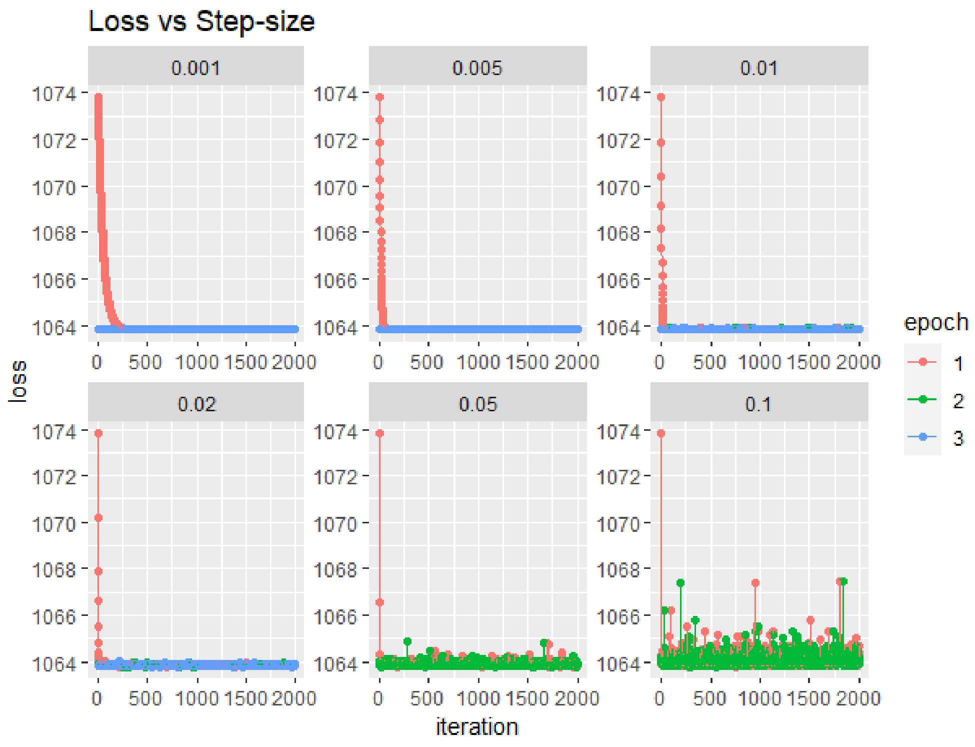
The main part of the SGD is the same in the basic one and in the variants. The max.iter parameter is used to control the number of epochs. Tolerance gap is also implemented for parameter tuning versions to check for convergence. Then, in each epoch, the index set is randomly permuted and a shuffled.order array is created. At each iteration, gradient is evaluated at a point and model parameters adjusted accordingly. A constant step size is used for all variants. For the averaging variant, the average is calculated as in the second part of the below formula to reduce calculations.

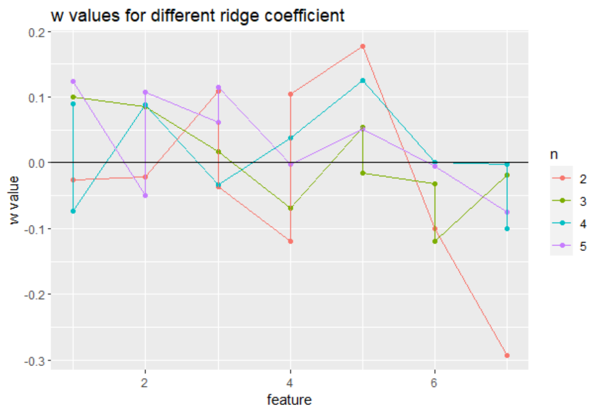
$$w_{t+1} = w_t - \eta \left[\frac{f_t'(w_t) - f_{\hat{t}}'(\phi_{\hat{t}})}{n} + \frac{1}{n} \sum_{i=1}^{n} f_i'(\phi_i) \right]$$

a. Coding Process and Parameter Tuning

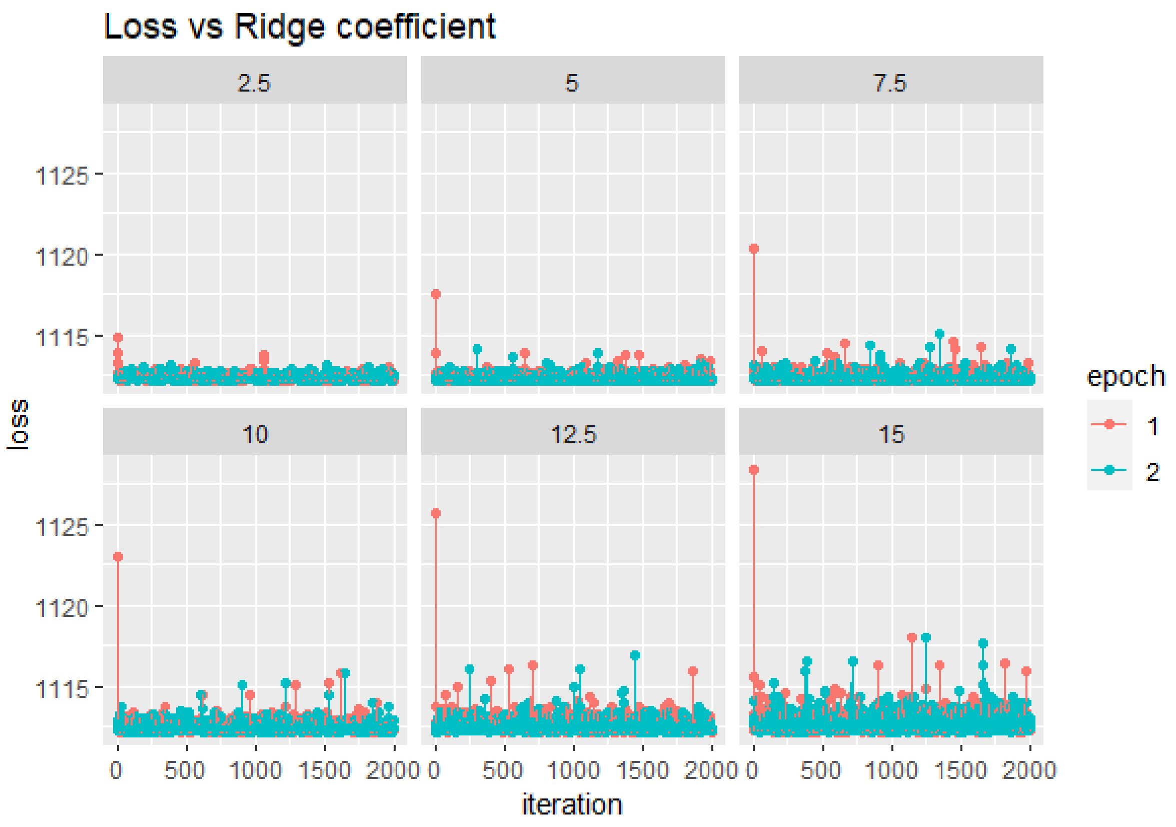
For logistic regression, there are 3 parameters: ridge coefficient, step size and momentum coefficient. As a stopping criteria for parameter tuning, average loss over epochs is used and if average has not improved with a percentage equal to the tolerance, the algorithm stops.

The first two are required for all SGD variants and tuned on the basic SGD. The small step size requires more iterations until convergence. The higher ones reduce loss quickly but may not converge in certain cases, may miss the local optima, may result in zig zag behaviour and therefore can cause large variance. Also, as step size increased, more epochs needed for convergence as seen in the graph below. As a result, the step size is selected as 0.01.

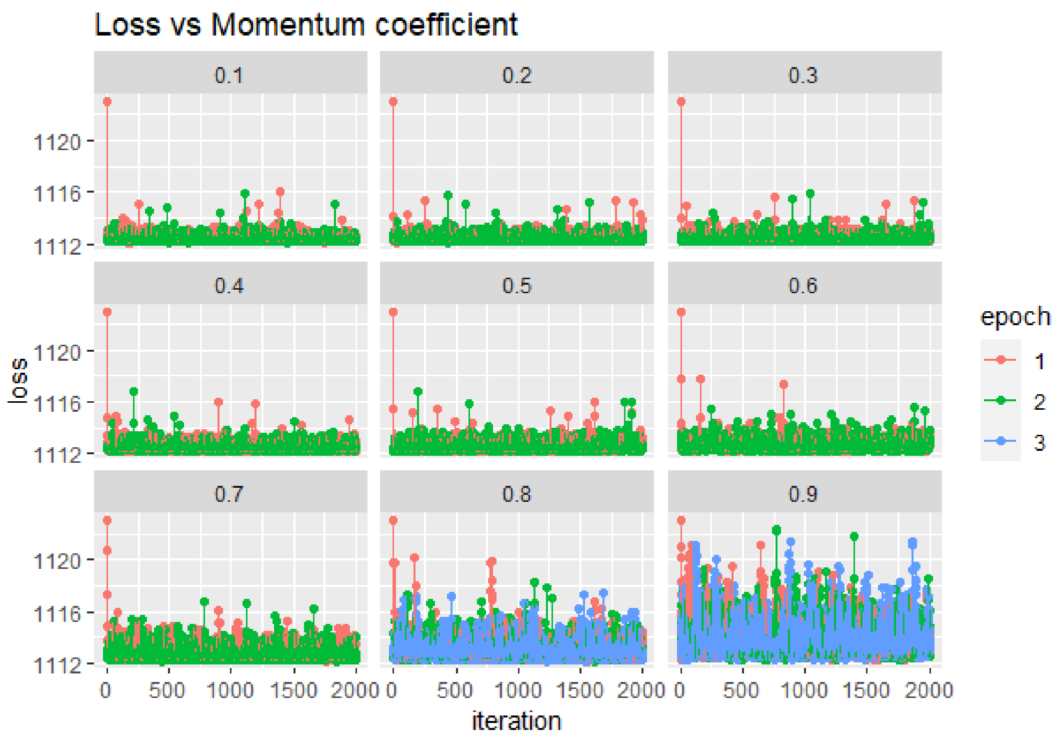




When training the basic SGD model with ridge penalty term on the loss function, increased penalty terms lead to reduced values of w towards 0 as expected. All ridge penalty values lead to convergence but high values of it resulted in higher variance. Ridge coefficient is set to 5 after inspecting graphs.



Lastly for the momentum coefficient, 9 different values are examined. Momentum coefficient determines the effect of previous gradient on the current gradient. So, as it gets higher, the current update heavily depends on the past information which in turn may result in missing local optima and zigzag behaviour. This behaviour can be seen in the graph below. After inspecting the graph, the 0.1-0.3 range seemed reasonable and the momentum coefficient was selected as 0.2.



b. Results for Data Set 1

The first data set classes are imbalanced. As this imbalance was not addressed in the data preprocessing or in the modelling stages, the results were as expected. The SGD and SGDM models quickly converged without considering the minority class and predicted all labels as the “-1” class.

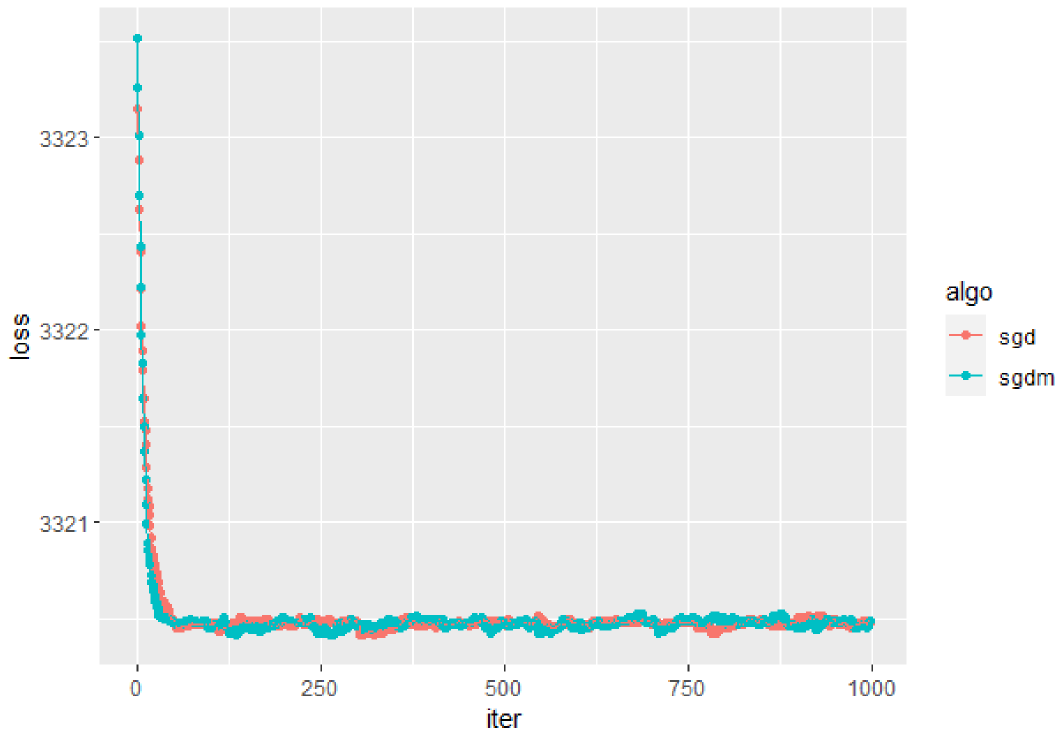
SGD	-1	1
-1	1937	0
1	63	0

SGDM	-1	1
-1	1937	0
1	63	0

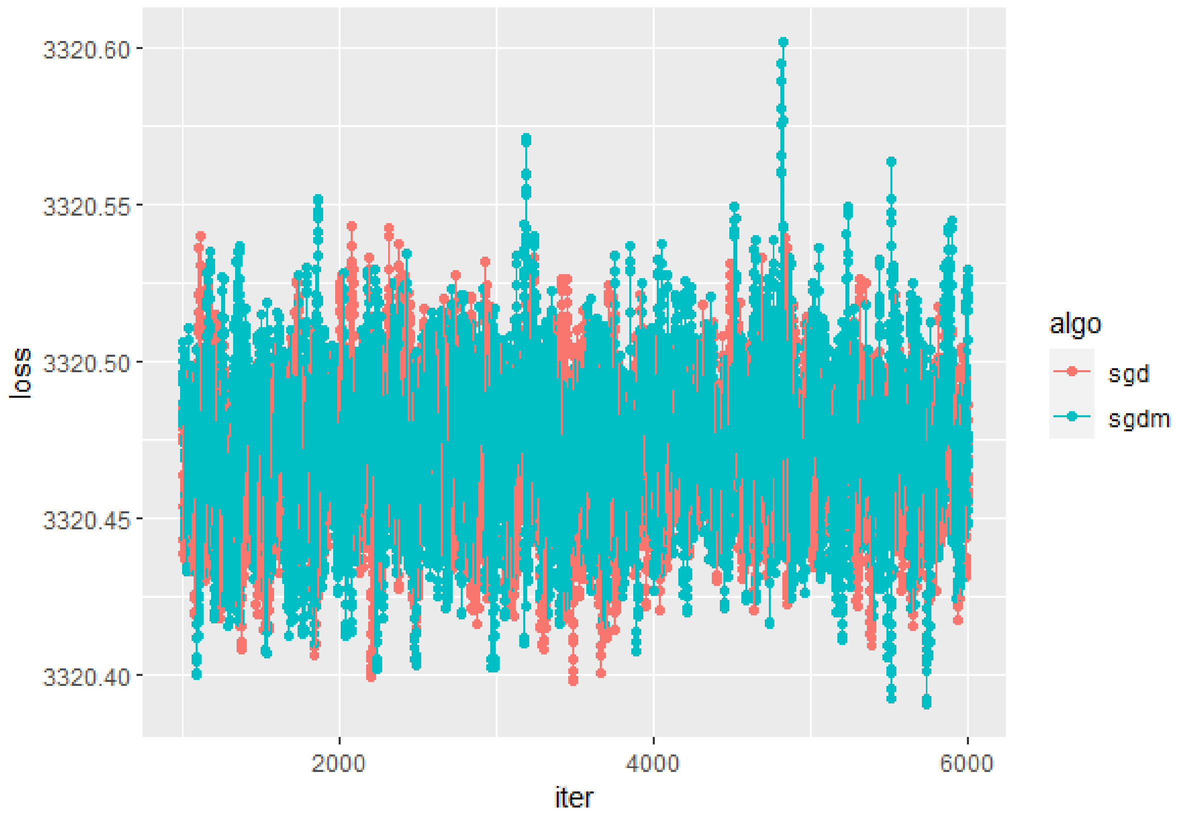
SAG model behaved differently as it did not predict all from the majority class. However, it did not converge probably due to parameter tuning and kept oscillating in all epochs. The confusion table for the SAG model is given below. Despite being worse in predicting the majority class, it was able to identify some minority class labels correctly. A good way can be predicting classes with both logistic regression with SGD/SGDM and SAG, and then comparing paired results.

SAG	-1	1
-1	971	46
1	966	17

The SAG model is excluded from algorithm comparison as it oscillates near a different level and complicates the graph. When SGD and SGDM are compared for the first 1000 data points, SGDM can be seen while converging quicker.



After convergence, the both algorithms keep oscillating around the same line as can be seen in the graph below. The graph shows the last 5000 iterations of the first epoch.



c. Results for Data Set 2

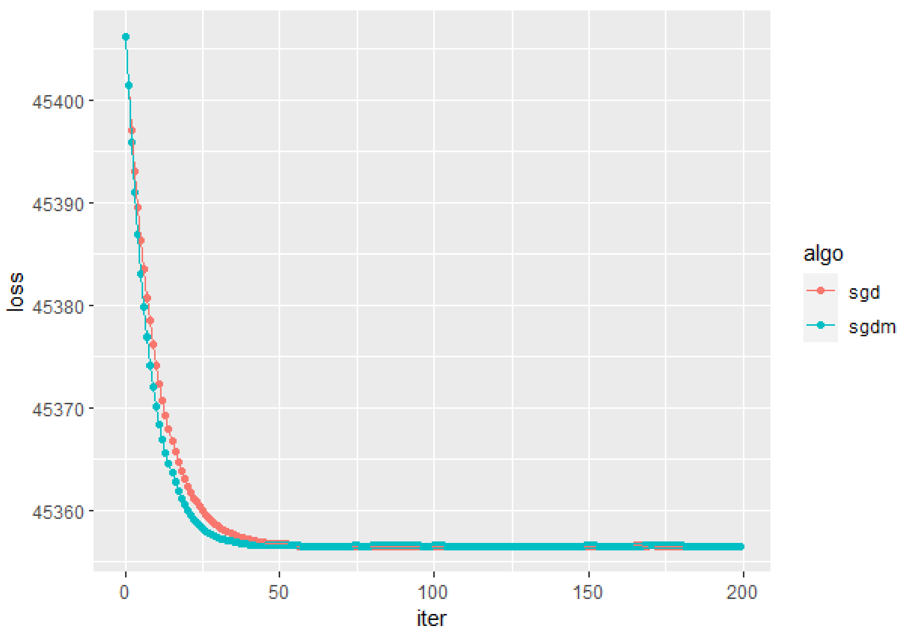
For the dataset 2, SGDM predicted all data points as class “-1” as in the first dataset. SGD predicted only 9 data points as class 1 and identified 6 of them correctly. Since SGD and SGDM converged quickly, even with more iterations they seem to perform more or less the same. The loss function of SAG continued to oscillate. Its results are worse than the first data set as it assumes two thirds of data belonging to the 11% minority class. This is a consequence of both oscillating and stopping criteria evaluation at the end of epochs. If stopping criteria is updated to check loss function value and oscillation behaviour at every iteration and if the algorithm stops at one of the minimum points of loss function, the results would get better.

SGD	**-1**	**1**
-1	**7342**	**3**
1	**887**	**6**

SGDM	**-1**	**1**
-1	**7345**	**0**
1	**893**	**0**

SAG	**-1**	**1**
-1	**2171**	**5174**
1	**396**	**497**

As both SGD and SGDM converged quickly, the plot shows only the first 200 iterations of both algorithms. Note that, since SAG oscillates around a different level, it is omitted again. In the graph below, SGDM can be seen as the winner again for convergence,



d. Results for Data Set 3

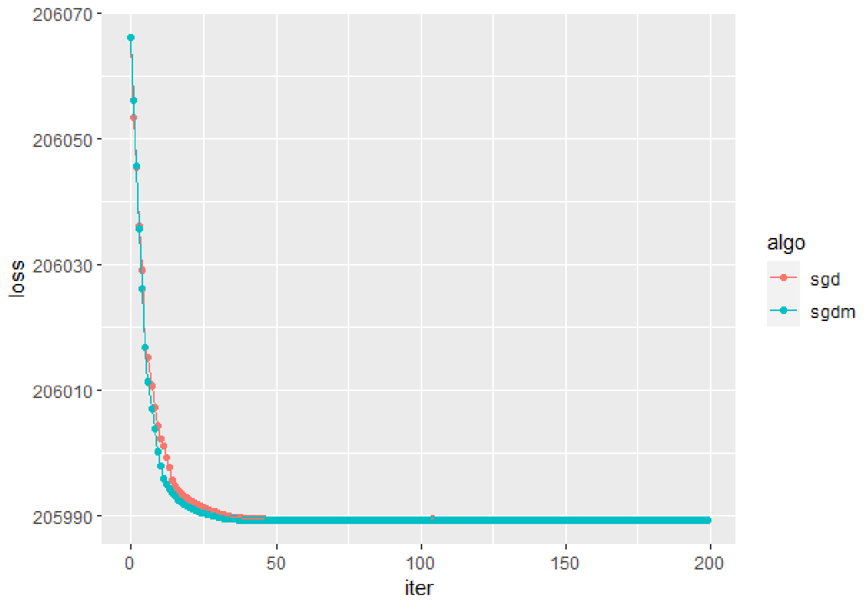
The basic version and variants of SGD performed closely on the third dataset. Data was balanced yet all algorithms lead logistic regression to favour the first class. SAG loss function continued to oscillate around another level again. However, w values with SAG provides the most true positive and true negative numbers, so the highest accuracy.

SGD	-1	1
-1	18739	14
1	21215	32

SGDM	-1	1
-1	18617	136
1	21149	98

SAG	-1	1
-1	18451	305
1	20805	442

When we compare the SGD and SGDM, we can see that SGDM still converges the fastest as in the first two dataset.



3. Support Vector Machine (SVM) with hinge loss and L2 regularization using stochastic subgradient algorithm

a. Coding Process and Parameter Tuning

In this part SVM approach is conducted using Jupyter Notebook. Html and .ipynb files are provided. Several functions are created to be used for all data sets. The following equation is used as the loss function, the aim is to decrease loss.

$$LOSS = \frac{1}{2}||w||^2 + C \sum_{(x,y)\in S} max\{0,\ 1 - y\widehat{y}\} \text{ where } \widehat{y} = sign(w^T x + w_0)$$

For the training part 2 functions are written, namely *find_subgradients* and *stochastic_subgradient_descent.* The second one calls the first in the body.

Find_subgradients function takes a feature dataframe, a target dataframe, a w vector, a w_0 value and C parameter as inputs. w_0 is the bias term in $(w^T x + w_0)$,if $w_0 = 0$ then the line passes from origin. This function returns the subgradients with respect to w and w_0. Parameter C will be tuned on the validation set.

Stochastic_subgradient_descent function takes a feature dataframe, a target dataframe, a C parameter, a subset proportion and the number of iterations as inputs. Here subset proportion is needed because at each iteration a different subset of data will be used to compute subgradients. Computing subgradients using the entire train data at each iteration takes a very long time. If this proportion is 1 then the entire data is used, if it is 0.2 then at each iteration 20% of data is selected again and subgradients are computed according to them. This function returns the optimal w, w_0 and loss array. Loss array includes all the losses at each iteration and it will be used to plot the improvement in loss. In the body of this function, some initial weights(w) and an initial bias (w_0) is selected. At each iteration a random subset of data is taken and subgradients are calculated using the find_subgradients function which was mentioned in the previous paragraph. Then weights(w) and bias(w_0) terms are updated according to the subgradients and step size. Step size is the reciprocal of the iteration number. This means the first step is 1, the second is ½, the third is ⅓, and so on. At each iteration predictions are calculated by the formula $(w^T x + w_0)$ and loss is calculated using the hinge loss function. This function is applied on the train set with a fixed C parameter.

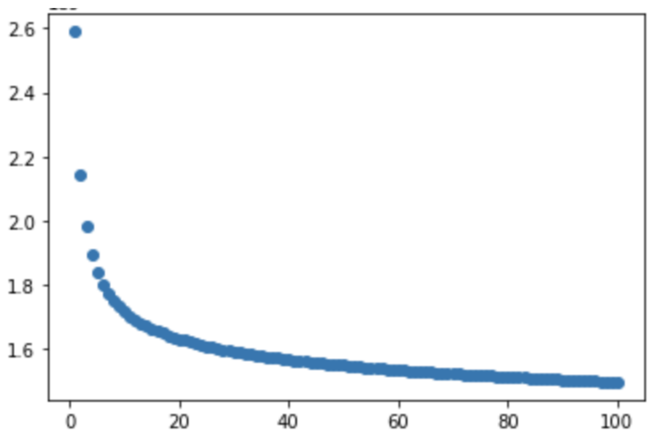
For the validation part *C_with_min_validation_loss* and *stochastic_subgradient_descent_with_best* functions are written. The first one is for tuning the C parameter. Values used for tuning are {0.05, 0.1, 0.3, 0.5, 0.7}. This function is applied on the validation set with w and w_0 found in the training part and the result is the optimal C parameter. The second function is a version of *Stochastic_subgradient_descent* which does not keep the loss values at each iteration. This function is written in order to decrease the computational effort. Using train set and best C parameter, this function returns optimal w, w_0 .

For the test part, *testing* function is written. This function takes features, actual targets and optimal w, w_0 values. It computes the predicted classes and returns a confusion matrix and the accuracy value. This part is applied on train, validation and test sets in order to observe overfitting.

For the second SVM approach quadratic kernel can be used. This corresponds to feature transformation. This transformation is done by *PolynomialFeatures* function from *sklearn.preprocessing* library. Then the actual features with no degree and interaction are dropped, and the models are tried on the transformed train validation test sets.

b. Results for Data Set 1:

Using the subset proportion as 1 to compute subgradients with a fixed C parameter ½ , 100 iterations resulted in the loss function improvement which is on the right.



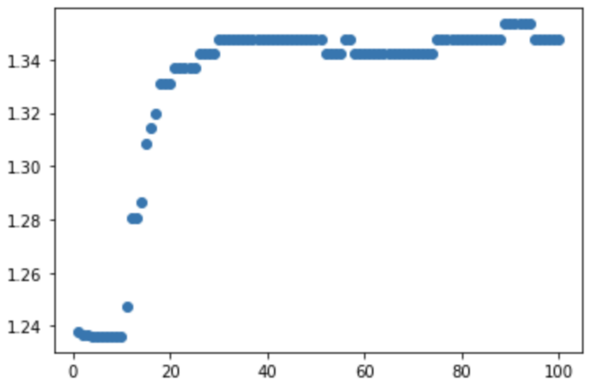
Tuning the parameter C on the validation set gave the best C parameter as 0.05. Then using it on the train set, optimal w, w_0 values are obtained and used on the test set. Accuracy values for the test-validation-train sets are 0.9665, 0.9656, 0.9670. They may seem nice but since the data is imbalanced, confusion matrices need to be checked. All the predictions are made as -1 which means no failure. So the model is not good at detecting the failures. There may be underfitting because train failures are also not detected so memorization did not happen, and the model did not learn.

TRAIN SET		
Predicted / Actual	-1	1
-1	5794	0
1	206	0

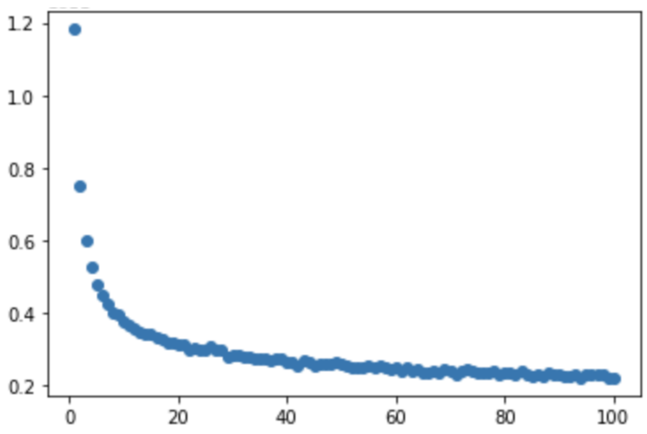
VALIDATION SET		
Predicted / Actual	-1	1
-1	1934	0
1	66	0

TEST SET		
Predicted / Actual	-1	1
-1	1933	0
1	67	0

For the second approach, after transforming the data sets, the same flow is conducted. This resulted in the training loss plot shown on the right side. It can be interpreted that the first data set is not suitable for kernel approach. So no validation and testing made here.



c. Results for Data Set 2:



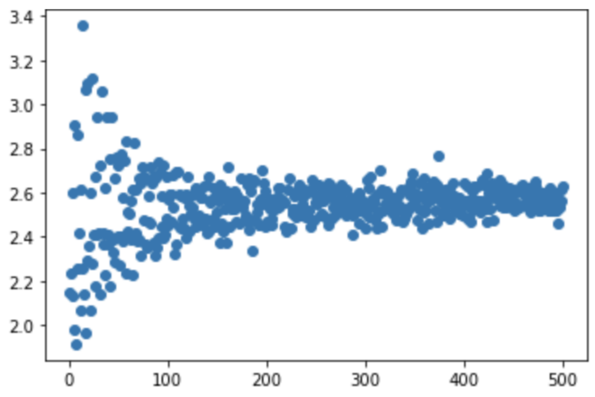
Using the subset proportion as 0.5 to compute subgradients with a fixed C parameter ½ , 100 iterations resulted in the loss function improvement which is on the right.

Tuning the parameter C on the validation set gave the best C parameter as 0.05. Then using it on the train set, optimal w, w_0 values are obtained and used on the test set. Accuracy values for the test-validation-train sets are 0.9065, 0.9015, 0.8999. Confusion matrices are also fine and test accuracy is higher than train accuracy, this indicates no overfitting.

TEST SET		
Predicted Actual	-1	1
-1	7254	82
1	688	214

VALIDATION SET		
Predicted Actual	-1	1
-1	7210	97
1	727	204

TRAIN SET		
Predicted Actual	-1	1
-1	21628	277
1	2157	650



For the second approach, after transforming the data sets, the same flow is conducted. This resulted in the training loss plot shown on the right side. This plot is strange since the loss is not decreasing but still it's converging. After the validation step and finding the best parameters, accuracy values for the test-validation-train sets are found as 0.8998, 0.9018, 0.8935. They are very similar to the non kernel approach.

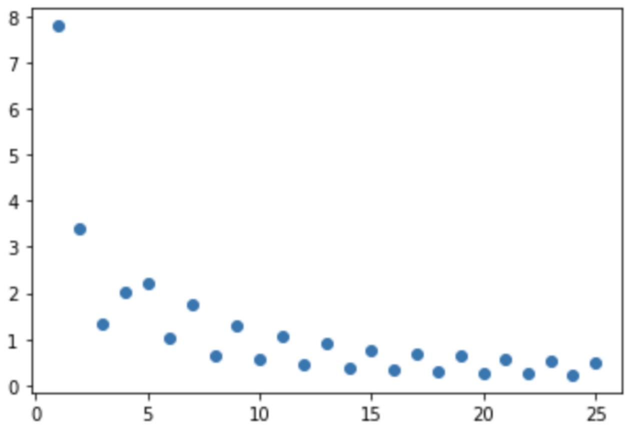
TEST SET		
Predicted Actual	-1	1
-1	7123	213
1	612	290

TRAIN SET		
Predicted Actual	-1	1
-1	21296	609
1	1817	990

VALIDATION SET		
Predicted Actual	-1	1
-1	7080	227
1	650	281

d. Results for Data Set 3:

Using the subset proportion as 0.2 to compute subgradients with a fixed C parameter ½ , 25 iterations resulted in the loss function improvement which is on the right.



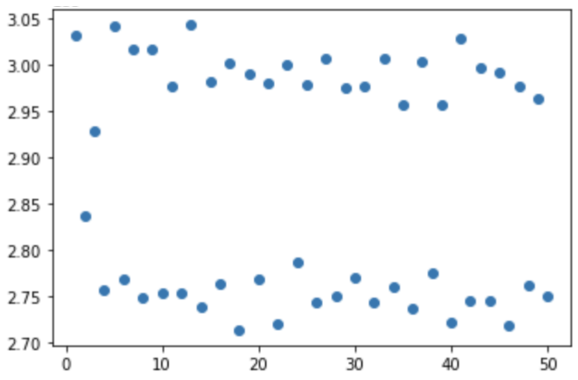
Tuning the parameter C on the validation set gave the best C parameter as 0.05. Then using it on the train set, optimal w, w_0 values are obtained and used on the test set. Accuracy values for the test-validation-train sets are 0.5784, 0.5769, 0.5774. The model learning is not good. So the linear SVM approach may not be suitable for this data set.

TEST SET		
Predicted / Actual	-1	1
-1	7393	11319
1	5543	15745

TRAIN SET		
Predicted / Actual	-1	1
-1	22953	34457
1	16312	47178

VALIDATION SET		
Predicted / Actual	-1	1
-1	7368	11469
1	5435	15728

For the second approach, after transforming the data sets, the same flow is conducted. This resulted in the training loss plot shown on the right side. This plot is strange and very bad since the loss is not decreasing and also not converging. After the validation step and finding the best parameters, accuracy values for the test-validation-train sets are found as 0.8998, 0.9018, 0.8935. They are very similar to the non kernel approach.



TEST SET		
Predicted / Actual	-1	1
-1	4204	14508
1	1758	19530

TRAIN SET		
Predicted / Actual	-1	1
-1	12596	43914
1	4977	58513

VALIDATION SET		
Predicted / Actual	-1	1
-1	4108	14729
1	1677	19486

4. Comparison of the Models

For the first data set, logistic regression trained with SGD and SGDM, predicted all classes as -1 for the test set. SAG performed worse in terms of accuracy but predicted a small amount of class 1 correctly. SVM with non kernel and kernel approaches did not give good results. Since all the predictions are -1 for train, validation and test sets. This indicates underfitting since learning is not enough. In conclusion, the relationship between the features and the target may not be linear.

For the second data set, SGD and SGDM predicted all labels as -1. SAG performed very poorly as it incorrectly predicted most of the data coming from the minority class. SVM with non kernel and kernel approaches gave good results. Train, validation and test accuracies are near 90% and the percentage based confusion matrices are very similar for both models. This indicates no overfitting so both of the SVM approaches are suitable for this data set.

For the third data set, all logistic regression approaches provide similar results. Contrary to the previous case, SAG achieved the most accuracy rate in this dataset. SVM with non kernel and kernel approaches did not give good results, all the accuracy metrics for train, validation and test sets are near 57% for both models. So SVM approaches may not be a good fit to this data set.

5. References

[1] Jason Brownlee, Gradient Descent With Momentum from Scratch, https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/

[2] Sebastian Ruder, An overview of gradient descent optimization algorithms, https://ruder.io/optimizing-gradient-descent/

[3] CS760 Machine Learning Project, Variance Reduction Methods, http://pages.cs.wisc.edu/~spehlmann/cs760/_site//project/2017/05/04/intro.html

[4] Support Vector Machines, Ryan M. Rifkin, https://www.mit.edu/~9.520/spring08/Classes/class05.pdf

[5] B. Schölkopf and A. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, 2001.

[6] Machine Learning: A Probabilistic Perspective, K.P. Murphy, MIT Press, 2012

[7] IE585 Machine Learning and Optimization, Figen Öztoprak Kaya