# Classifying New Particle Formation

Group 128, Viktoria Lilitskaia

2025-12-03

## Introduction

Check the ***Github Repo*** for the source code and additional plots.

Kaggle submissions were made using the same name as stated in the document subtitle (Viktoria Lilitskaia).

Special information about the project: we started as a team of three people, but, unfortunately, my two teammates had to leave the course. Initially, we agreed on using Python and its libraries such as numpy, pandas, etc. to do the project, but now I am switching to R, because I am more familiar with it in data science context. However, some preliminary work such as exploratory data analysis was done using Python in Jupyter Notebook, and this is why the Github repository contains both `.Rmd` and `.ipynb` project files. Now, the primary programming language is R and `project.Rmd` is the main project source code file.

### Custom functions

Under this section, we will define custom functions that are used in this project and provide their code snippets.

#### Perplexity

```
perplexity <- function(p, true) {
  exp(-mean(log(p)))
}
```

#### Alpha and lambda selection for binary glmnet

This function will be used to automatically perform selection of optimal value of $\alpha \in \{0, 0.5, 1\}$ and $\lambda$ based on the smallest MSE from 10-cross-fold validation using `cv.glmnet`. This is used in selection of models for performing multiclass logistic regression.

```
alpha.lambda <- function(data.x, y) {
  X <- model.matrix(~ ., data = data.x)[, -1]

  foldid <- sample(1:10, size=length(y), replace=TRUE)
  alphas <- c(1, 0.5, 0)
  results <- data.frame(alpha = alphas, lambda.min = NA, cvm.min = NA)

  for (i in seq_along(alphas)) {
    cv <- cv.glmnet(X, y, alpha=alphas[i], nfolds=10, foldid = foldid)
```

```
    results$lambda.min[i] <- cv$lambda.min
    results$cvm.min[i] <- min(cv$cvm)
  }

  best <- results[which.min(results$cvm.min), ]

  list(alpha = best$alpha, lambda = best$lambda.min)
}
```

### Binary logistic accuracy and perplexity estimation

This function is used to assess the accuracy and perplexity of a logistic classification method in a binary setting by using random train and validation subsets of the data J times (we chose J=100 in all cases) and collecting the resulting perplexity and accuracy scores into one vector.

```
logistic.acc.perp <- function(data, J, y, lambda) {
  n <- floor(0.5*nrow(data))
  acc.binary.lr <- numeric(J)
  perp.lr <- numeric(J)

  for (i in 1:J) {
    picked <- sample(seq_len(nrow(data)), size = n)

    train <- data[picked, vars]
    valid <- data[-picked, vars]
    y.valid <- y[-picked]

    X.train <- model.matrix(~ ., data = train)[, -1]
    y.train <- y[picked]

    model <- glmnet(X.train, y.train, family="binomial", alpha=1, lambda=lambda)

    valid.matrix <- model.matrix(~ ., data=valid)[, -1]
    valid.pred <- predict(model, newx=valid.matrix, type="response")
    pred <- ifelse(valid.pred >= 0.5, 1, 0)

    acc.binary.lr[i] <- mean(pred == y.valid)
    perp.lr[i] <- perplexity(valid.pred)
  }

  list(acc.binary.lr, perp.lr)
}
```

### Multiclass logistic

This function performs multiclass logistic regression using given parameters for a `glmnet` model and returns the most probable class for the test data as well as the full matrix of class probabilities.

```
logistic.multiclass <- function(train.x, train.y, test, alphas, lambdas, classes) {
  results <- matrix(data=NA, nrow=nrow(test), ncol=length(classes))

  X <- model.matrix(~ ., data = train.x)[, -1]
```

```r
  test.matrix <- model.matrix(~ ., data=test)[, -1]

  for (i in seq_along(classes)) {
    fit <- glmnet(X, train.y[, i], family="binomial", alpha=alphas[i], lambda=lambdas[i])
    pred <- predict(fit, newx = test.matrix, type="response")
    results[, i] <- pred
  }

  # Index of max probability per row
  max_id <- max.col(results, ties.method = "first")
  # Assign class
  predicted <- classes[max_id]
  return(list(prob = results, predicted = predicted))
}
```

# Preprocessing

As a part of preprocessing the data, we need to remove unnecessary columns such as `date`, `id`, and `partlybad`. We also need to encode our class values as numeric values. First, when dealing with binary classification, we encode *nonevent* as 0 and *any event* as 1.

In a multiclass setting, preprocessing of response values can vary depending on the method used. For example, multinomial regression that we use later requires classes to be parsed through a `factor` function to make them categorical. I assume this method to be a default in the most multiclass classification methods in R.

Later, in preprocessing for multiclass logistic regression for "one-vs-all" one class out of four was encoded as 1 and the rest as 0.

## Normalization

We can also normalize our data, because generally normalizing yields more stable results in some classification algorithms such as SVMs or logistic regression. This however can depend on an algorithm, but we can try classification both with and without normalization and compare the results.

# Data Exploration

This task was done using Python and the Seaborn library.

Our main task in data exploration was to produce pairplots of variables in the data set to study possible collinearity issues and get more understanding of data overall. However, there is a problem in studying pairplots: we have 100 variables! To deal with this problem, we decided to plot pairplots of 5 variables at a time. Unfortunately, this reduced the number of predictors we can view at the same time, but was useful in the sense of readability. As a result, we produced in total of 20 pairplot images which are saved as `.png` files in the `plots` folder in the Github repository.

We also made a mistake in code once, which produced a pairplot image of 20 variables at the same time. It was unintended, but we saved the image in the `plots` folder anyway, because it gave some broader insight into the variables.

As we see from the generated images, there is a large amount of features that are highly linearly correlated, some of them producing strictly linear plots. We will have to address this issue in the feature selection process.


Figure 1: An example of a pairplot produced by Seaborn. You can find more examples in the "plots" folder.

## Pearson Correlation Coefficient

We can construct a correlation matrix between all variables to further detect correlations.

```
##
##    CO2336.mean     CO2336.std     CO242.mean      CO242.std    CO2504.mean
##              1              1              2              2              3
##     CO2504.std       Glob.std    H2O168.mean    H2O336.mean     H2O336.std
##              3              1              4              5              1
##     H2O42.mean     H2O42.std    H2O504.mean    H2O504.std    H2O672.mean
##              6              2              7              3              8
##     H2O672.std     H2O84.mean     H2O84.std       NET.mean       NET.std
##              4              9              5              2              3
##     NO336.mean     NO336.std     NO42.mean     NO504.mean     NO504.std
##              1              3              3              4              7
##     NO672.mean     NO672.std     NO84.mean      NO84.std    NOx168.mean
##              6              8              8             11              8
##    NOx336.mean    NOx336.std    NOx42.mean    NOx504.mean    NOx504.std
##              9              1             10             11              2
##    NOx672.mean    NOx672.std    NOx84.mean     NOx84.std      O342.mean
##             12              3             13              4              1
##       O342.std     O3504.mean     O3504.std     O3672.mean     O3672.std
##              1              2              2              3              3
##      O384.mean      O384.std       PAR.mean       PAR.std     RGlob.mean
##              4              4              4              5              6
##      RGlob.std RHIRGA168.mean  RHIRGA168.std RHIRGA336.mean  RHIRGA336.std
##              7              7              7              9              7
##  RHIRGA42.mean   RHIRGA42.std RHIRGA504.mean  RHIRGA504.std RHIRGA672.mean
##              9             12             10              8             11
##  RHIRGA672.std  RHIRGA84.mean   RHIRGA84.std      RPAR.mean       RPAR.std
##              8             14             18              2              3
##        SWS.std      T168.mean       T168.std       T42.mean        T42.std
##              1              6             18              7             20
##      T504.mean       T504.std      T672.mean       T672.std       T84.mean
##              8             20              9             21             10
##        T84.std      UV_A.mean       UV_A.std      UV_B.mean       UV_B.std
##             22             24             30             28             29
```

In this table, we see how many times a variable appears in a pair of highly correlated predictors whose Pearson correlation is $> 0.7$. There are too many correlated pairs to print them all at once, so we decided only to print their counts.

## Feature selection

As noticed before, the data has significant amount of collinear variables, making a lot of features useless for classification. We can use different methods to detect problems further. For the time being, we decided to use feature selection method already embedded in model training algorithm later, such as Lasso/Ridge implemented in `glmnet` library to be used in logistic regression scenario. We will assess the most suitable feature shrinkage method by comparing different values of $\alpha$ used in `glmnet`.

## Binary classification: logistic regression

In this section, we will compare different models for binary classification using logistic regression and comparing different settings. Logistic regression was chosen because it is a solid baseline classification approach that is easy to implement and analyze, as well as having built-in methods for variable selection such as in `glmnet`.

### Setup

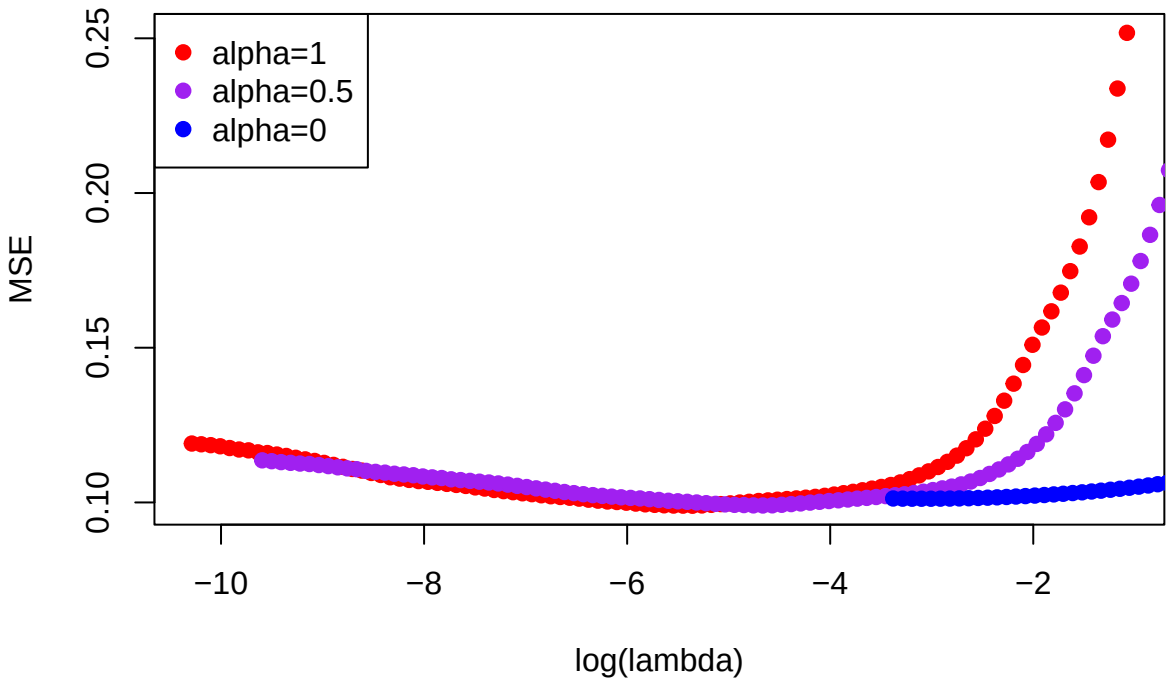
Because standard `glm` has convergence issues on the full dataset, I will be using `glmnet` library and experiment with different values of $\alpha$ to see what gives the best accuracy on the validation set. I will be using k-fold cross validation to measure each model's squared loss and mean absolute error to choose the best-performing model. I will choose `k=10` and CV implementation by the R library `glmnet`. I will try values of $\alpha = 0$, 0.5 and 1.

I will use both scaled and unscaled data sets to see which one gives better performance.

```
##                : Matrix
```

```
## Loaded glmnet 4.1-10
```

We can plot MSE based on values of $\log \lambda$ and choose the best value of $\alpha$.




According to the plot, the absolute lowest MSE is reached by $\alpha = 1$. We can also compare the lowest values of MSE by code:

```
## [1] "Lowest MSE when alpha=1:"
```

```
## [1] 0.09893267
```

```
## [1] "Lowest MSE when alpha=0.5:"
```

```
## [1] 0.09901761
```

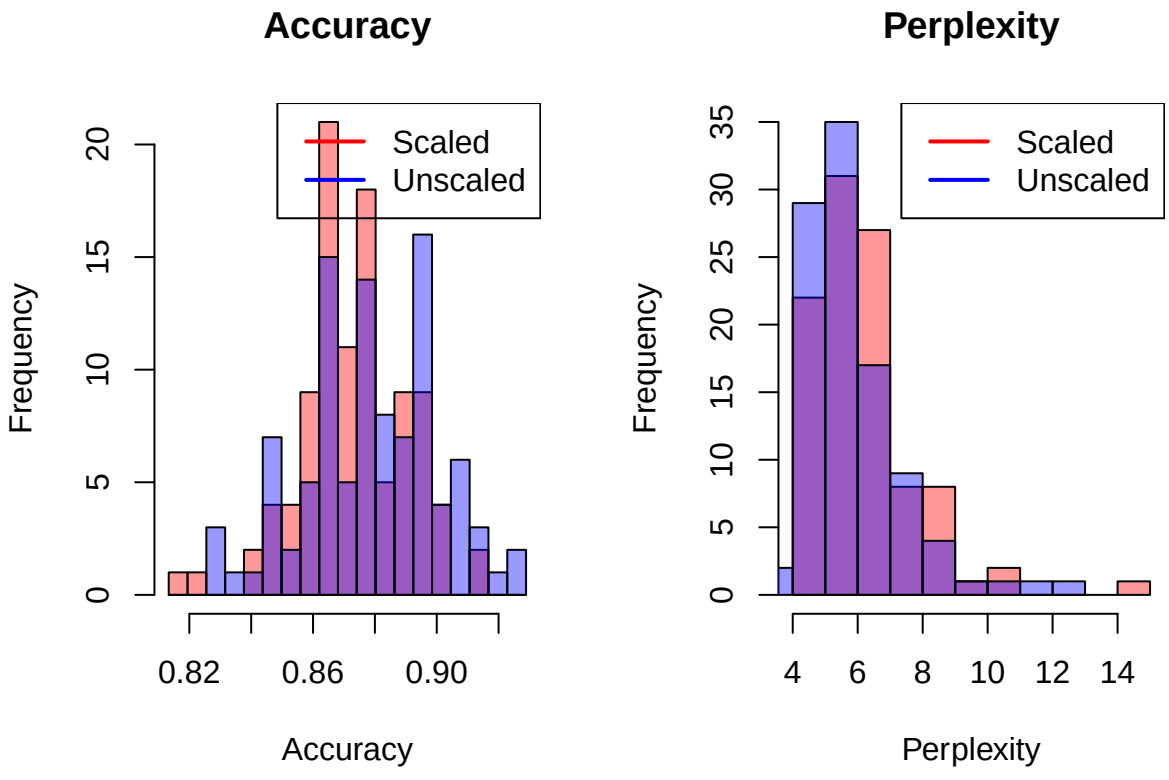
```
## [1] "Lowest MSE when alpha=0:"
```

```
## [1] 0.1011944
```

So, to fit our model, we choose $\alpha = 1$ and the lambda that gives the minimum MSE in `cv1`.

### Accuracy and perplexity

We can estimate the accuracy of our classification by choosing a number of random validation sets and training the model with selected parameters on the rest of the training set. I will run the code to select a random training (50%) and validation (50%) sets and estimate the accuracy as well the perplexity, using our custom defined `perplexity` function. By the end of it, we get the following distribution of accuracy and perplexity scores:




Mean and standard deviation values of accuracies between scaled and unscaled data:

```
## [1] "Mean accuracy and std for scaled data:"
```

```
## [1] 0.8729778
```

```
## [1] 0.01783582
```

```
## [1] "Mean accuracy and std for unscaled data:"
```

```
## [1] 0.8788444
```

```
## [1] 0.02185209
```

Based on the mean and standard deviation values and histogram comparison, we notice that scaling the data produces slightly smoother distribution of accuracies, while unscaled data seems to be more susceptible to random fluctuations. Nonetheless, average accuracy looks similar between both data sets.

# Multiclass classification

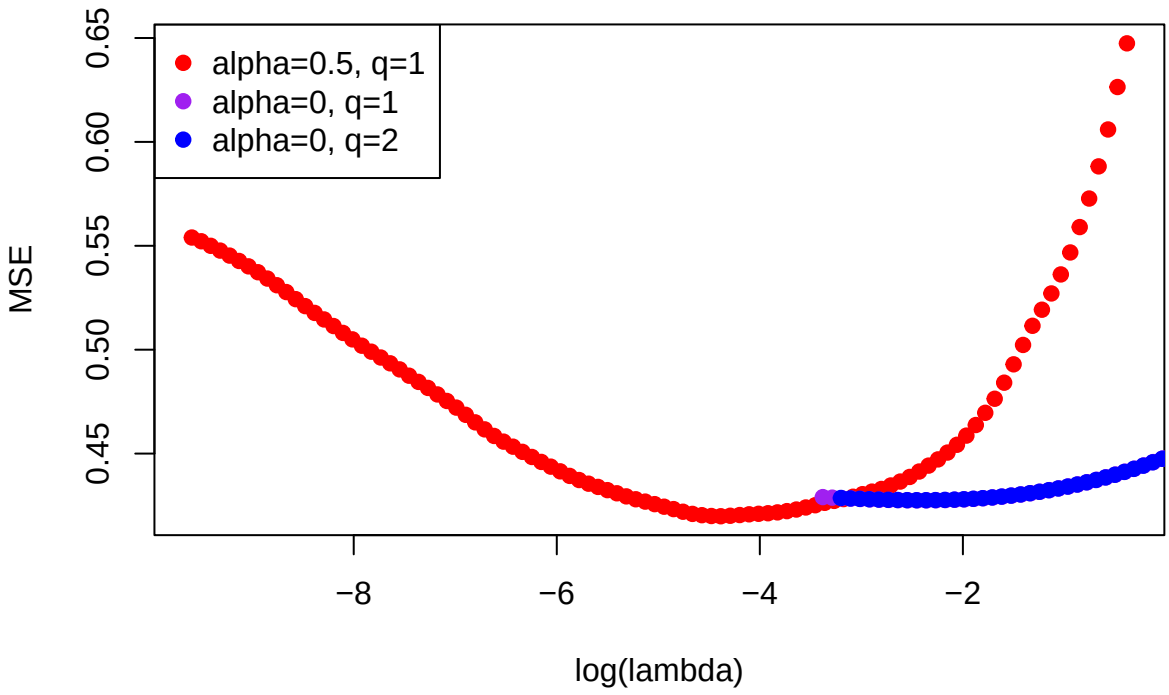
There are different ways to approach multiclass classification. One can utilize a ready-to-run multinomial algorithm such as multinomial family in the `glmnet` library that can deal with multiple classes. We will test this option first, because it is the most simple to setup.

## Multinomial regression

Additionally to the parameter $\alpha$, the multinomial regression has a parameter $q \in \{1, 2\}$. $q = 1$ means lasso penalties for each parameter and $q = 2$ means a grouped lasso penalty on all the coefficients of a variable. We will try both values and assess which one works better for our data set using `cv.glmnet` as in the binary classification task with $K = 10$, as well as testing different values of $\alpha \in \{1, 0.5, 0\}$. We will use the scaled and unscaled data sets and compare the performance on both of them.

After running the code for both values of q and all three values of alpha, I've had convergence issues with $\alpha = 1, q = 1$, $\alpha = 1, q = 2$, $\alpha = 0.5, q = 2$. So, we will remove these models from the test and compare the rest which did not have convergence issues.

Now, we can plot MSE based on values of $\log \lambda$ and choose the best value of $\alpha$ and $q$.




```
## [1] "Lowest MSE when alpha=0.5, q=1:"
```

```
## [1] 0.4198375
```

```
## [1] "Lowest MSE when alpha=0, q=1:"
```

```
## [1] 0.4274888
```

```
## [1] "Lowest MSE when alpha=0, q=2:"
```

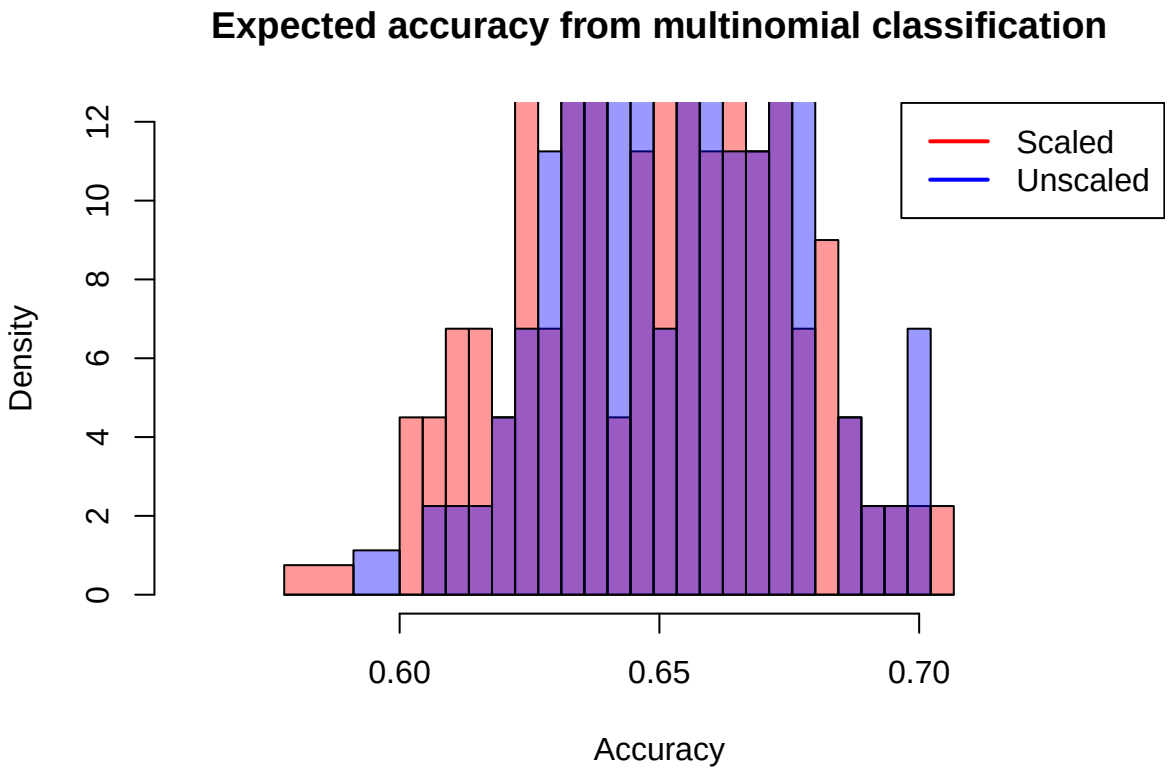
```
## [1] 0.4275621
```

Based on these criteria, we will choose parameters $\alpha = 0.5, q = 1$.

Next, we can see the resulting accuracy distribution for scaled and unscaled data:

```
## Warning in lognet(x, is.sparse, y, weights, offset, alpha, nobs, nvars, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
## Warning in lognet(x, is.sparse, y, weights, offset, alpha, nobs, nvars, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
## Warning in lognet(x, is.sparse, y, weights, offset, alpha, nobs, nvars, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
```

```
## Warning in lognet(x, is.sparse, y, weights, offset, alpha, nobs, nvars, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
## Warning in lognet(x, is.sparse, y, weights, offset, alpha, nobs, nvars, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
```




Mean and standard deviation values of accuracies between scaled and unscaled data:

```
## [1] "Mean accuracy and std for scaled data:"
```

```
## [1] 0.6516
```

```
## [1] 0.02377846
```

```
## [1] "Mean accuracy and std for unscaled data:"
```

```
## [1] 0.6526667
```

```
## [1] 0.023504
```

We can expect our multinomial classification accuracy to be on average between 0.62 and 0.68. Distributions of accuracies are also pretty similar between scaled and unscaled data; scaled data has a very slight advantage by being more centered near the average accuracies, while unscaled data generate wider range of accuracies especially at the lower end.

## Logistic regression

Another possible solution is to train several models using binary logistic regression and make them "fight" against each other either in one-vs-one or one-vs-all settings.

We can setup our logistic regression to give predictions in a multiclass setting. We can use either "one-vs-all" or "one-vs-one" strategy. For this problem, we chose "one-vs-all", because it requires less amount of models to train (four models instead of six), and also does not require to split the training data into chunks of only two classes. We will test this approach on scaled and unscaled data.

In this strategy, we will train four separate logistic regression models using `glmnet`: *nonevent vs. rest*, *Ia vs. rest*, *Ib vs. rest*, *II vs. rest*. We will use the same strategy as in binary classification problem to choose the appropriate value of $\alpha$ for each model. After that, we will run our test data set through all four models and assign class with the largest probability.

### Parameter selection: scaled data

#### Nonevent vs. rest

This one is similar to the binary classification of nonevent vs. event, but in this case our model will give the probability of *nonevent* instead of probability of *event*.

```
## [1] "Nonevent vs. rest: alpha=1 lambda=0.00325079498372315"
```

#### Ia vs. rest

```
## [1] "Ia vs. rest: alpha=0 lambda=0.371929176832423"
```

#### Ib vs. rest

```
## [1] "Ib vs. rest: alpha=1 lambda=0.0102051474654586"
```

#### II vs. rest

```
## [1] "II vs. rest: alpha=1 lambda=0.0181152016501647"
```

```
## Warning in lognet(x, is.sparse, y, weights, offset, alpha, nobs, nvars, : one
## multinomial or binomial class has fewer than 8 observations; dangerous ground
```

### Parameter selection: unscaled data

```
## [1] "Nonevent vs. rest: alpha=1 lambda=0.00325079498372315"
```

#### Ia vs. rest

```
## [1] "Ia vs. rest: alpha=1 lambda=0.00316048095928025"
```

#### Ib vs. rest

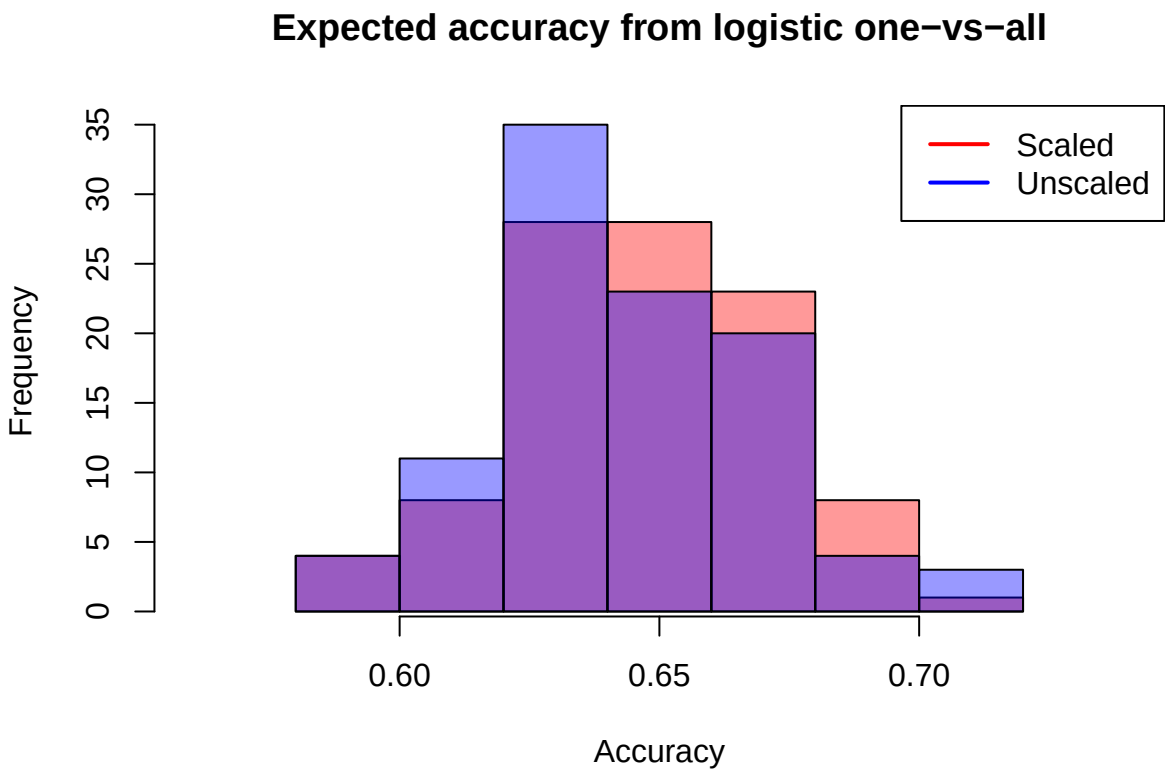
```
## [1] "Ib vs. rest: alpha=0.5 lambda=0.0224002734900308"
```

#### II vs. rest

```
## [1] "II vs. rest: alpha=1 lambda=0.0198814114477135"
```

**Accuracy estimation**

Next, we will perform similar comparison as in the previous chapters, and plot the distributions of accuracies between scaled and unscaled data:




Mean and standard deviation values of accuracies between scaled and unscaled data:

```
## [1] "Mean accuracy and std for scaled data:"
```

```
## [1] 0.6488
```

```
## [1] 0.02506989
```

```
## [1] "Mean accuracy and std for unscaled data:"
```

```
## [1] 0.6441778
```

```
## [1] 0.0247482
```

Based on the histograms and mean accuracies, we see how the scaled data tend to produce better accuracy than the unscaled. The difference is more clear in logistic one-vs-all setting compared to multinomial regression.

## Results

| Model | Bin.accuracy | Mult.accuracy | Perplexity | Kaggle.score |
|---|---|---|---|---|
| Scaled LR | 0.8729778 | NA | 6.183173 | 0.74878 |
| Unscaled LR | 0.8788444 | NA | 5.852265 | 0.74705 |
| Scaled LR (one-vs-all) | NA | 0.6488000 | NA | NA |
| Unscaled LR (one-vs-all) | NA | 0.6441778 | NA | 0.74360 |
| Scaled Multinomial | NA | 0.6516000 | NA | 0.74878 |
| Unscaled Multinomial | NA | 0.6526667 | NA | 0.74705 |

After trying both scaled and unscaled data on Logistic Regression and Multinomial Regression, we noticed that its effect on classification accuracy differs between classification method. It is generally expected that logistic regression gives better predictions with scaled data, but in the case of binary logistic regression and multinomial regression, the difference was not noticeable. However, in these cases scaling the data lead to generally smoother distribution of accuracy, which can especially be seen in histograms of binary LR accuracies. The resulting differences are more considerable in case of multiclass logistic in one-vs-all setup, where average accuracy was noticeable higher with scaled data. As such, we can conclude that in this particular problem, it is preferable to scale the data, because it introduces no disadvantages and in some cases makes predictions slightly more accurate and stable.

Other than that, we notice that the quality of predictions does not differ significantly between different settings of logistic and multinomial regressions. We decided to focus on optimizing these models, because logistic regression is a good baseline classification method, and we were interested to see if this method could be optimized to give results comparable to more complex models. We consider this experiment to be a success; despite only using simple models, we managed to get pretty good private score for Kaggle that guaranteed us 11th place in the competition.

Another thing that could possibly have been done is to also try other classification machine learning methods such as KNN, SVM or Random Forest, and compare the results with those from our baseline logistic solution, but alas, we ran out of time.

## Self-grading report

If I was to suggest a grade for my term project, I would give myself a grade of 3. I think I managed to produce a clear and suitable report which goes into detail of what I have done in this project and why certain choices were made. The report could have been more polished, however, and I could have done more work in comparing my model of choice with models of other families. I believe that my work shows some understanding, although the depth of it is not for me to decide, since I would not possibly notice the limits of my knowledge by myself. I think the reason of my choice of the classification method might be a little weak; Firstly, I chose logistic regression just to try something baseline to analyze the data because of the method's simplicity, but later I decided to stick with it and instead try to optimize the baseline methods as much as I could. I also feel like there was no problem in following the schedule; I was not late for any submissions, although I did not manage to implement all the classification methods I wanted to try, but this might be a problem just with planning too much initially. I also believe that my deliverables follow the instructions given: they include reasoning for my choices, there is no unexplained code dumps, visualizations are mostly clear, etc.