Big Data Science
Exam Project

Michael Pieters and Weijia Feng

May 23, 2019

1 Task 1: Michael Pieters

1.1 Research Question

The goal is to classify cells into their corresponding cycle phases (Interphase, Prophase, Metaphase, Anaphase, Telophase), using the information present in the 213 features extracted by imaging flow cytometry.

1.2 Data Preparation

There were no missing values in the supplied dataset. Since many features can only contain positive values (area shapes of cells), spurious values like zeroes need to be handled with care. If there are features that contain a large portion of zeroes (more than 10 percent), then these are considered bad features for the given problem and will be removed. Only one feature (minimum intensity of the image) contained a significant part of zeroes (28993 in total) and was removed. Such a feature has almost no variation in values and the algorithms that will be applied, will not learn anything from it. If features contained a small proportion of zeroes, then these were imputed by the median of their corresponding class.

1.3 Metrics

An imbalance occurs when one or more classes (minority class) have very low proportions in the data as compared to the other classes (majority class). Mostly in these situations, the main interest is in correctly classifying the minority class. Accuracy is a really bad metric in such a situation. High accuracy scores can be obtained by always predicting the majority class.

A better metric is the F-score which conveys the balance between precision and sensitivity. The measure is 0 when either the precision or the sensitivity is 0. Low precision means lots of negative results (majority class) being labeled as positive (minority class). Low sensitivity means lots of positive results being labeled as negative.

The formula for this measure is given by:

$$F = \frac{2 \times sensitivity \times precision}{sensitivity + precision} \tag{1}$$

In our cases we have 5 different classes. An equally weighted F-score should be used which means that every class individually is considered the positive class and metrics are calculated by comparing this positive class to all the rest of the classes. This can be done by setting **average** equal to *macro* for the f1 score function in scikit-learn.

Another metric is the AUC under the ROC. The ROC curve calculates the sensitivity and specificity across a continuum of cutoffs. An appropriate balance can be determined between sensitivity and specificity (how well it doesn't miss any true negatives) using the curve. AUC is simply the area under the ROC curve. An area of 1 represents a perfect model and an area of 0.5 represents a worthless model.

When you have a data imbalance between positive and negative samples, it is preferred to use F1-score because ROC averages over all possible thresholds! It is reported that for imbalanced data AUC can still give values above 0.8, due to the large amount of false positives with respect to true positives. AUC will be reported for all the methods used but will not be given much weight in deciding which is the best model for this particular dataset.

A criticism of the F-score is that it does not focus on true negatives. Although that is somewhat remedied by making use of the weighted F-score where each class is considered the positive class for each weight. In this case correct predictions for each class is taken into account.

To handle this criticism, one can make use of Cohen's Kappa which takes into account accuracy that would be generated by chance.

$$kappa = \frac{\text{total accuracy} - \text{random accuracy}}{1 - \text{random accuracy}} \tag{2}$$

$$\text{total accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$\text{random accuracy} = \frac{(TN + FP)(TN + FN) + (FN + TP)(FP + TP)}{(TP + TN + FP + FN)^2} \tag{4}$$

1.4 Feature Selection

Several feature selection methods are available to select the most useful features in your dataset. One of the first feature selection methods we will explore is the removal of collinear features. Our data is highly multi-collinear as you can see in Figure (1) where the correlation matrix is plotted. The condition number of the correlation matrix is a staggering $1.09e + 16$.

Data scientist Will Koehrsen offers the public an open source code (ref. 8) for which collinear features can be detected based on a specified correlation coefficient value. For each pair of correlated features, it identifies one of the features for removal. With a correlation threshold of 0.6, 162 features are removed (50 features remain) and the condition number of the correlation matrix is reduced to 32.65.

A different feature selection method is based on Random Forest feature importance. When training a decision tree, it can be computed how much each feature decreases the weighted impurity in the tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure. When the dataset has a lot of correlated features, any of these correlated features can get a high feature importance. But once one of them is used, the importance of others is significantly reduced since the impurity they can remove, is already removed by the first feature. A random forest with 20 trees is used to determine the feature importance values of our dataset. If the feature importance is lower than 0.01, the feature is removed. In the end, only 27 features remained.

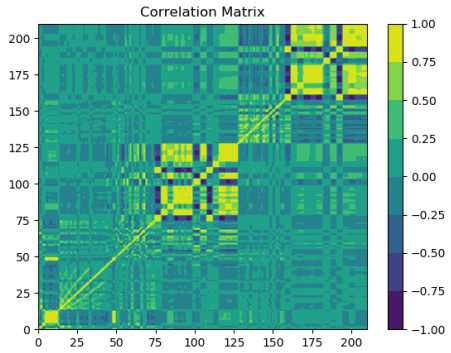

Figure 1: Correlation matrix

There is criticism that the scikit-learn Random Forest feature importance strategy is biased. To get more reliable results, one has to make use of permutation importance which comes with the rfpimp package. More information can be found in ref. 7. Only 8 features scored a non-zero feature importance. (By keeping only those features, model performance dropped significantly. No further results through this feature selection technique will be published.)

These feature selection methods will be employed to increase training speed, model's interpretability and model's performance.

1.5 Model Training

The dataset is split into a training set and validation set. Cross validation will be employed on the training set to tune the model's hyperparameters based on the metric Cohen Kappa. With these optimal hyperparameters, F-score, AUC and Cohen Kappa are calculated on the validation set. This will give us an idea how well the model performs and if it under- or overfits. A better approach would be nested cross-validation but because of time constraints this technique is not utilized. Only five folds will be taken for stratified cross validation to ensure that every class is represented in each fold. To speed up model training, the majority class (Interphase) is randomly downsampled to 1000 instances.

The following four models will be employed: Logistic Lasso Regression, Random Forest, Linear Discriminant Analysis and a Neural Network with two hidden layers. Parameters that will be tuned are the regularization parameter C for Logistic Lasso Regression, maximum depth of the tree, maximum amount of features, minimum samples at each leaf and minimum samples at each split for Random Forest, the amount of components for Linear Discriminant Analysis and activation function, regularization parameter α and the size of the hidden layers for the Neural Network.

1.6 Improvements

One way to further deal with class imbalance is to make use of penalized learning algorithms that increase the cost of classification mistakes on the minority classes. For logistic regression this can be done by setting **class weight** to *balanced*. The saga-solver needs to be used for l1-regularization and unfortunately the coefficients do not converge with balanced class weights, even in the case of setting the maximum number of iterations for the optimization to a fairly high number (but still reasonable).

Another way to combat class imbalances is to generate synthetic samples for the minority classes. The most popular of such algorithms is called SMOTE or the Synthetic Minority Over-sampling Technique. As its name suggests, SMOTE is an oversampling method. It works by creating synthetic samples from the minor class instead of creating copies. The algorithm selects two or more similar instances (using a distance measure) and perturbing an instance one attribute at a time by a random amount within the difference to the neighboring instances. An implementation of SMOTE can be found in the imblearn library.

Special care has to be taken when implementing SMOTE while doing cross validation. SMOTE only has to be used on the training folds, so that no information leaks to the test folds with risk of overfitting.

1.7 Results

The results of all methods can be found in Table 1. SMOTE has been applied for all methods. For feature selection either multicollinear feature removal or random forest feature importance is used. Methods will be compared with eachother based on their metric scores on the holdout validation set.

Based on Cohen Kappa the best methods are a Random Forest (100 trees) with multicollinear feature removal and a Neural Network with feature importance. If you take F1-score into consideration the Random Forest with multicollinear feature removal does best, followed by Logistic Regression and a Neural Network, both with feature importance. Logistic Regression gives the best AUC scores. Every metric gives a slightly different picture of what model performs best. Since the Random Forest with multicollinear feature removal performs best on both Cohen Kappa and F1-score, this is chosen as our best performing model. Judging the Cohen Kappa metric on the cross-validated dataset and the validation sample, the model slightly underfits.

Table 1: Model Performance

Method	Feature Selection	Optimal Parameters	CK (CV)	CK (Val)	F1 (Val)	AUC (Val)
Logistic Regression	Multicollinear	$C = 0.18$	0.654	0.652	0.727	0.880
Logistic Regression	Feature Importance	$C = 0.02$	0.605	0.629	0.755	0.899
Random Forest	Multicollinear	max depth = 20 max features = auto min samples leaf = 1 min samples split = 2	0.714	0.737	0.776	0.852
Random Forest	Feature Importance	max depth = 20 max features = auto min samples leaf = 1 min samples split = 5	0.663	0.701	0.700	0.813
LDA	Multicollinear	components = 1	0.599	0.612	0.733	0.860
LDA	Feature Importance	components = 1	0.570	0.558	0.685	0.836
Neural Network	Multicollinear	activation = relu $\alpha = 10$ layer sizes = (10, 9)	0.690	0.631	0.677	0.820
Neural Network	Feature Importance	activation = relu $\alpha = 10$ layer sizes = (10, 7)	0.675	0.733	0.751	0.852

2 Task 2: Weijia Feng

2.1 Visual inspection of the data

The distributions of the responses are highly skewed to the left. After the log-transformation, the response variables have an approximate symmetric distribution. Figure (2) shows three out of 12 responses. The results of the rest of responses are comparable to Figure (2).

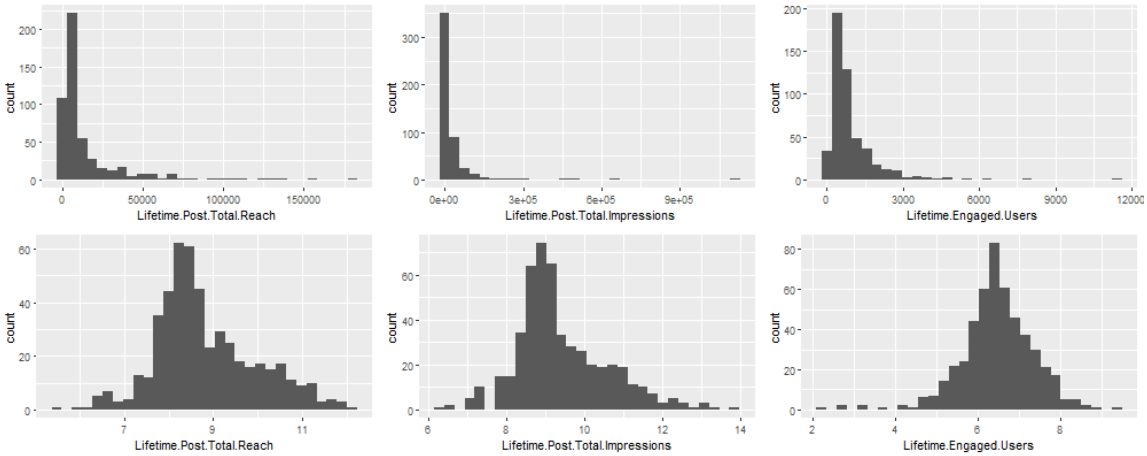

Figure 2: Upper panel: histograms of response variables. Lower panel: histograms of log-transformed response variables.Results of 3 from 12 response variables. From left to right: post total reach, post total impression, engaged users.

Figure (3) shows the frequency of each level for all categorical predictors. Figure (4) shows all pairwise plots of predictors. It can be seen that the distribution of the continuous predictor *Page total likes* is quite different for different months (Figure 4. 18).

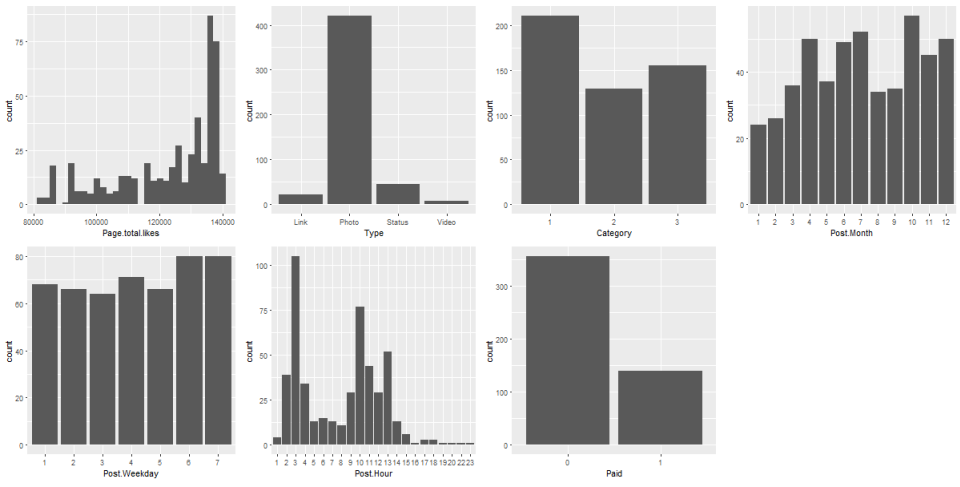

Figure 3: Histogram of the *Page total likes* (upper left). Frequencies of levels for each categorical predictor.

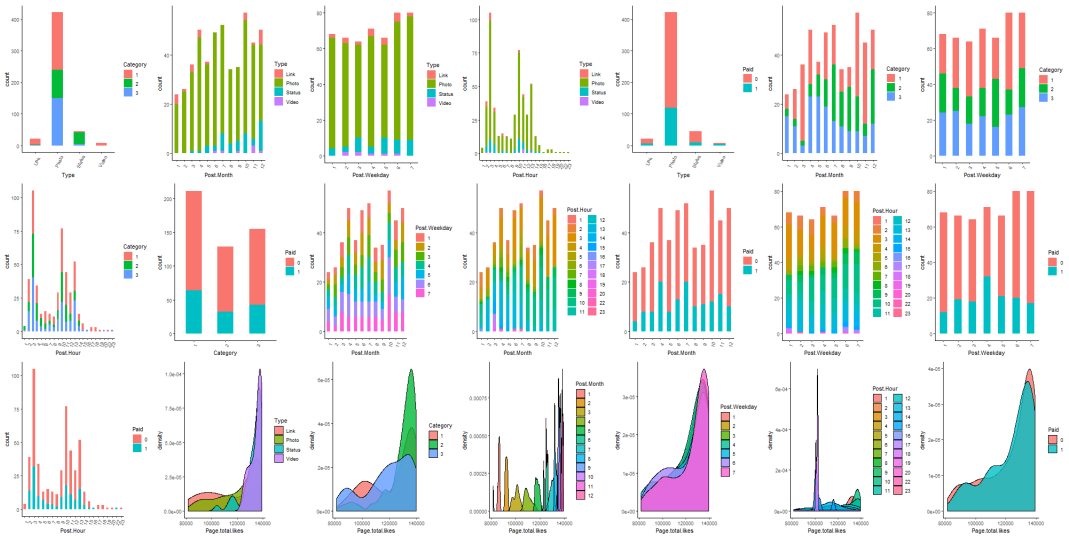

Figure 4: Pairwise plots of all predictors. From upper left to lower right: 1: type by category. 2: Type by month. 3: Type by weekdays. 4: Type by hours. 5: Type by paid. 6: category by month. 7: category by weekdays. 8: category by hours. 9: category by paid. 10: weekdays by month. 11: hours by month. 12: paid by month. 13: hour by weekdays. 14: paid by weekdays. 15: paid by hours. Density of page total likes by type (16), by category (17), by month(18), by weekdays(19), by hours(20), by paid(21).

2.2 Predictive Results

2.2.1 Lifetime people who have liked a page and engaged with a post

The minimal average mean absolute percent error (MAPE) is 45.7 % as achieved by the random forest (number of tree set to 1000). The mean R squared value is 0.2 for the random forest. LASSO with a linear kernel regression had a similar performance based on 5-fold cross validation (CV). See table 2.

Table 2: Prediction summary of *Lifetime people who have liked a page and engaged with a post*.RMSE: root mean squared error. Rsq: R squared. MAE: mean absolute error. MAPE: mean absolute percent error.

Method	RMSE	Rsq	MAE	MAPE
Random Forest	368.719	0.204	226.045	45.725
LASSO	382.542	0.162	234.695	48.545
ANN	389.234	0.15	244.908	50.922
KNN	389.615	0.132	239.09	47.173
SVM	391.529	0.177	234.18	46.993

Another analysis was done by splitting the data into a training set and a testing set. The training set consists of 80 % of data and the testing set consists of 20 % of the data. The predicted responses using the LASSO model and observed responses are plotted in figure 5a. The importance as measured by the increase of note purity (as assessed by the residual sum of squares) for each splitting variable is plotted in figure 5b. *Page total likes*, *Type*,*Paid*,and some timing variables are most important splitting variables. The following predictors are with largest coefficients for the LASSO: for *Type*: status: 0.2,link: -0.12, video: -0.07. *Month*: Nov: -0.08, Dec: -0.08, *Page total likes*: -0.07.

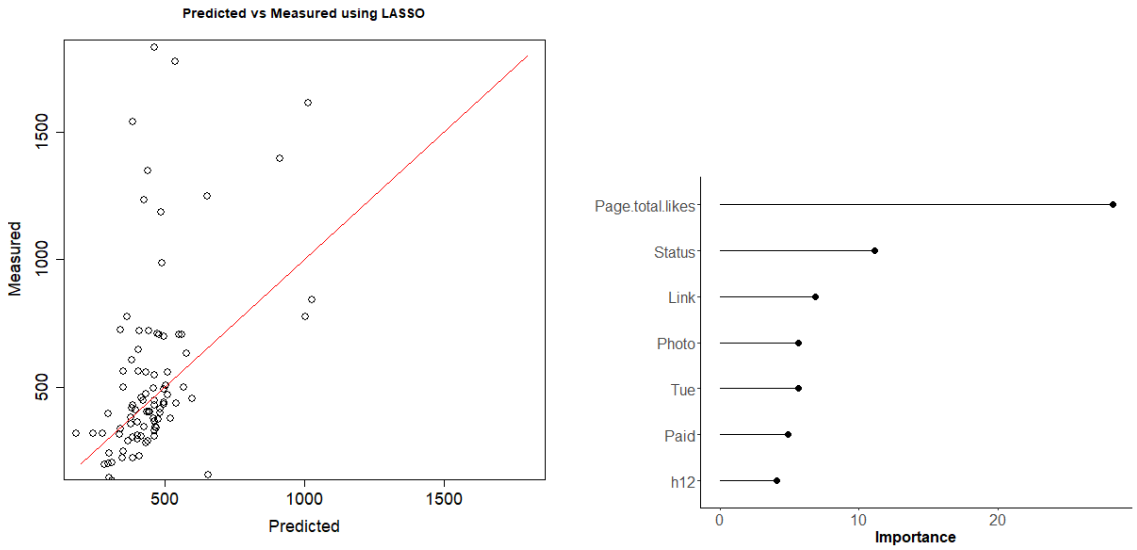

Figure 5: a(left): Predicted versus observed responses for the *Lifetime people who have liked a page and engaged with a post* using the LASSO. b(right): Importance of splitting variables for the random forest model.

2.2.2 All response variables

The best prediction performance based on minimal RMSE for each of 12 responses was summarized in table 3. Apart from the response variable of *Lifetime.Post.Consumers*, random forest was the model that produced lowest RMSEs among all 5 models. Feed-forward artificial neural network (ANN) produced the lowest RMSEs for the variable *Lifetime.Post.Consumers*. Table 4 shows the model with lowest MAPE. The prediction of the *Lifetime.Post.Consumers* and the *Lifetime people who have liked a page and engaged with a post* are with an MAPE of below 50 %.

Table 3: Model with lowest RMSE for each response. Order of responses is according to column order of the data.

Response	Best Model	RMSE	Rsq	MAE	MAPE
1	Random Forest	16075.41	0.106	8115.121	79.746
2	Random Forest	25322.71	0.142	12676.59	73.6
3	Random Forest	636.593	0.255	378.802	50.408
4	ANN	564.136	0.346	332.022	55.796
5	Random Forest	958.457	0.243	551.61	54.205
6	Random Forest	14889.05	0.097	7468.684	72.012
7	Random Forest	7422.883	0.102	3930.44	70.833
8	Random Forest	368.719	0.204	226.045	45.725
9	Random Forest	11.198	0.014	5.906	112.256
10	Random Forest	151.658	0.048	90.312	81.683
11	Random Forest	23.159	0.107	14.218	69.316
12	Random Forest	180.957	0.05	108.737	77.548

Table 4: Model with the lowest MAPE for each response. Order of responses is according to column order of the data.

Response	Best Model	RMSE	Rsq	MAE	MAPE
1	SVM	17128.26	0.03	8644.246	77.83
2	SVM	27710.86	0.021	13639.09	71.259
3	Random Forest	636.593	0.255	378.802	50.408
4	Random Forest	569.518	0.346	315.337	47.805
5	Random Forest	958.457	0.243	551.61	54.205
6	SVM	15469.86	0.073	7616.034	65.172
7	SVM	7794.997	0.067	4033.992	64.631
8	Random Forest	368.719	0.204	226.045	45.725
9	LASSO	11.285	0.019	5.994	111.301
10	Random Forest	151.658	0.048	90.312	81.683
11	Random Forest	23.159	0.107	14.218	69.316
12	Random Forest	180.957	0.05	108.737	77.548

2.3 Discussion and conclusion

Generally, the predictions of the each of the 12 response variables based on 7 predictors are not accurate. Identification of outliers would be necessary to improve the prediction accuracy. Data transformations like creation of dummy variables would be necessary so that algorithm like k nearest neighbors can be implemented. For the prediction of *Lifetime people who have liked a page and engaged with a post*, it can be seen that the variables of *Page total likes*, *Type* are with more predictive power than the other variables. This could be because of some causal effect between those predictors and the response. More predictors that are associated with the response would be needed to improve predictive performances.

2.4 Appendix Task 2

2.4.1 New manipulations and analyses

All 12 response variables are log-transformed. This is needed to define outliers in the way that is consistent with Moro et al. Outliers of a response is defined as those with the value of the range of median plus or minus 1.5 time the inter-quartile range. In addition, log-transformation may alleviate potential violations of the assumptions of the linear model, such as normally distribution and constancy of variance of errors. Final summaries were based on exponentiated predictions and observations.

R squared, mean absolute error and mean absolute percent error were added as measured of prediction performance.

2.4.2 Analyses not done or done in a different way

Histograms were made for continuous predictors and all responses. Pairwise bar plots were made between categorical predictors. Density of the continuous predictor were plotted by each level of the categorical predictor. Each dummy variable corresponds to one level of a categorical predictor, except for one dummy for the predictor hour. A dummy variable is defined to represent the hour between 16:00 to 01:00, as the count during this period is particularly sparse. A scatter plot of predicted versus the observed responses were used. Bagged tree model was not used because it is not covered in the course. All model parameter tunings were based on 5-fold cross validation.

3 Question 5: Big Data Analysis

3.1 Simple approach vs modern statistics (Weijia)

Simpler models like linear regression may have advantage in interpretebility, and efficiency with computation compared to more complex models like support vector machines, artificial neural networks (ANN) etc. On the other hand with the linear regression, the lineaer assumption and distributional assumption may limit the predictive power with complex problems. More complex models like ANN and random forest do not suffer from such problems.

3.2 Parallel Computing (Michael)

First of all, any algorithm where some function has to be optimized (Logistic Regression, LDA, Neural Network), can be parallelized by making use of mini batch gradient descent optimization. The training data is split over multiple devices. At each step, each local machine estimates the gradient using a subset of the data (mini batch). All gradient estimates are passed to a central machine, which aggregates them to perform a global parameter update.

A random forest which is actually an ensemble method which consists out of a group of Decision Tree classifiers, lends itself to parallelization as well. Each Decision Tree classifier individually can be run on a part of the training data on a single device. All the results are captured by a central machine which calculates the majority vote.

The process of cross-validation for finding optimal parameters can be parallelized in two different ways which can be combined. The gridsearch itself can be split into parameter subsets which can be run on multiple devices in parallel. Additionally is it possible to run each fold of the cross-validation on a separate machine.

To give an example, a pseudocode is included that shows how you can distribute a gridsearch for Ridge Regression over several servers (or machines) with Tensorflow.

```
import tensorflow as tf

cluster_spec = tf.train.ClusterSpec({
    "ps": [
        "127.0.0.1:2221",  # /job:ps/task:0
    ],
    "worker": [
        "127.0.0.2:2222",  # /job:worker/task:0
        "127.0.0.3:2223",  # /job:worker/task:1
        "127.0.0.4:2224",  # /job:worker/task:2
    ]})

task_ps0 = tf.train.Server(cluster_spec, job_name="ps", task_index=0)
task_worker0 = tf.train.Server(cluster_spec, job_name="worker", task_index=0)
task_worker1 = tf.train.Server(cluster_spec, job_name="worker", task_index=1)
task_worker2 = tf.train.Server(cluster_spec, job_name="worker", task_index=2)

with tf.device("/job:ps/task:0"):
    X = tf.constant(, dtype=tf.float32, name="X")
    y = tf.constant(, dtype=tf.float32, name="y")
    XT = tf.transpose(X)
```

```
with tf.device("/job:worker/task:0"):
    lambda_ridge = 0.1
    diagonal = [lambda_ridge]*p
    theta_0 = tf.matmul(tf.matmul(/
                        tf.matrix_inverse(tf.add(tf.matmul(XT, X), /
                                                 tf.matrix_diag(diagonal))), XT), y)

with tf.device("/job:worker/task:1"):
    lambda_ridge = 1
    diagonal = [lambda_ridge]*p
    theta_1 = tf.matmul(tf.matmul(/
                        tf.matrix_inverse(tf.add(tf.matmul(XT, X), /
                                                 tf.matrix_diag(diagonal))), XT), y)

with tf.device("/job:worker/task:2"):
    lambda_ridge = 10
    diagonal = [lambda_ridge]*p
    theta_2 = tf.matmul(tf.matmul(/
                        tf.matrix_inverse(tf.add(tf.matmul(XT, X), /
                                                 tf.matrix_diag(diagonal))), XT), y)

with tf.Session("grpc://127.0.0.1:2221") as sess:
    theta_0_value = theta_0.eval()
    theta_1_value = theta_1.eval()
    theta_2_value = theta_2.eval()
```

---

4 References

1. Chao Chen, Andy Liow & Leo Breiman, *Using Random Forest to Learn Imbalanced Data*, Department of Statistics, UC Berkeley

2. Nitesh Chawla, Kevin Bowyer, Lawrence Hall & Philip Kegelmeyer, *SMOTE: Synthetic Minority Over-sampling Technique*, Journal of Artificial Intelligence Research 16, 2002

3. Haibo He & Edwardo Garcia, *Learning from Imbalanced Data*, IEEE Transactions on knowledge and data engineering, vol. 21, no. 9, September 2009

4. Josephine Akaso, *Predictive Accuracy: A Misleading Performance Measure for Highly Imbalanced Data*, Paper 942, Oklahoma University, 2017

5. Tom Fawcett, *ROC Graphs: Notes and Practical Considerations for Data Mining Researchers*, Intelligent Enterprise Technologies Laboratory, January 2003

6. Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn & TensorFlow*, 1st edition, March 2017

7. `https://explained.ai/rf-importance/index.html`

8. `https://towardsdatascience.com/a-feature-selection-tool-for-machine-learning-in-python-b64dd23710`