Variable Screening in Machine Learning - Analysis Pipelines for a Metabolomic Dataset

Ngoc Duong (nqd2000)

Introduction

Alzheimer's Disease (AD) is a common and serious brain disorder that has damaging effects on memory and cognitive skills, especially among the elderly. In many cases of Alzheimer's disease, patients lose basic functions such as conversation as well as recognition of and responses to the external environment. This makes AD the 6th leading causes of death in the US [1]. Risk factors of AD include the natural process of aging, genetic predisposition, and lifestyle-related behaviors such as lack of exercise, poor diet, or lack of social connections and activities [14]. Since AD has no effective cure, prevention plays an important role in reducing the burden of this disease. Nowadays, advancement in high-throughput technologies and availability of multi-omics data, we are able to analyze the physio-pathological mechanisms of this complex disease at a molecular level.

The goal of this project is two-fold: 1) use classification tools in statistical learning to identify metabolites that are predictive of incident AD, and 2) during this process, explore potential analysis pipelines (with and without different variable screening steps) and observe the resulting behaviors of classification performance, as well as computation time reduction.

Data and Methods

Data: The data comes from PPG-DS study (Columbia University is a participating institution), in which patients were followed longitudinally over the duration of 9 visits (each visit 1-1.5 years apart). Untargeted plasma metabolomic data were obtained from 297 of the patients on their first visits. There are measurements on 2,708 metabolites' expression levels for each subject. The outcome is incident AD, which is obtained from clinical diagnoses (gold standard). Among 297 patients with available metabolomic data, 243 have clinical diagnoses. The distribution for the outcome variable is: 60 with incident AD and 183 without incident AD. Descriptive statistics of other baseline covariates are presented in **Table 1**.

Methods: The full pipeline I consider includes a univariate screen, followed by a multivariate screen, and classification model. I experiment removing either or both of the screening steps and compare predictive performance and computation time. The learners I consider are Elastic Net (Enet), Partial Least Squares (PLS), Random Forest (RF), Extreme Gradient Boosting (XGBoost), and Support Vector Classifier (SVC).

Potential Pipelines:

1) Univariate screen $\rightarrow$ Multivariate screen $\rightarrow$ Learner
2) Univariate/Multivariate screen $\rightarrow$ Learner
3) Learner on full data

Since there is class imbalance (only about 25% of subjects have incident AD), I will use both ROC AUC and precision-recall (PR) AUC as metrics and up-sample the minority class while training. PR curve can be more informative than a ROC curve in the case of imbalanced classes [13]. Unit-variance scaling and centering will be used to preprocess the design matrix. In the interest of time, models will be trained using 10-fold CV. Due to limited sample size, I decided not to create a hold-out set, but instead report the mean cross-validated ROC AUC (performance across test folds). Learners are trained using package caret.

More details on screening steps

Univariate screen: one option is to re-iterate two-sample t-test or its nonparamteric equivalents, the Mann-Whitney U-test or Krukal-Wallis test, over all metabolites across cases and controls. Here, I used linear model in package "limma", which computes the moderated t-statistic using empirical Bayes approach, adjusting for covariates age at blood draw, sex, ethnicity, and APOE4 allele presence. The moderated t-statistic can be seen as a mixture of frequentist and Bayesian approach since it uses the posterior variance in the formula for t-statistic. This also takes into account variability information from other metabolites, instead of considering the sample variance of each metabolite individually. This can result in more stable inference and low false-discovery rate [15]. Due to multiple testing, metabolites with FDR-adjusted p-values < 0.05 (under Benjamini-Hochberg procedure) are moved on to the next step (366), indicated by the green dots in the volcano plot (**Figure 1**).

Multivariate screen: PCA – I used the PCs that account for 90% of the total variance in the design matrix and fit PC-based RF and XGBoost. As we can see in **Figure 2**, the first two PCs from the whole set of metabolites do not differentiate the two classes well, while we can see some separation of these two classes along the direction of PC1 based on differentially expressed metabolites. Based on the "elbow" on the scree plot (**Figure 3**), I decided to use first four PCs for SVC. Important variables can be identified by calculating the total absolute loadings. Elastic net and Partial Least Squares were also considered at this step due to their fast implementation (features with non-zero coefficient for Enet and with highest Variable Importance Projection - VIP - for PLS) [4, 10].

Learner comparisons

Regularized regression: LASSO is a popular penalized regression method. Since LASSO uses L1-norm penalty, it might not perform well in the presence of multicollinearity (non-unique solutions) and can generally be unstable [7]. Another drawback is LASSO can only select up to p = n features which might not be attractive. Elastic net has both L1-norm and L2-norm penalty terms, which allows it to effectively shrink coefficients in a continuous fashion like ridge regression and also shrink some directly to 0 like LASSO. Unlike LASSO, it can select more than n features from p-dimensional input space. It can also select correlated variables, which is more suitable given interdependence between metabolites.

Enet has two tuning parameters: for α where 0 means Lasso and 1 means ridge, I tuned over the grid (0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 0.8). The shrinkage parameter λ was also tuned over a grid using cross-validation. Variable importance can be estimated as the sum of the beta coefficients in the different fitted models.

PLS is similar to PCR in the sense that it constructs linear combinations of the features, or map them on some latent space, and use these to predict the outcome. However, while PCs are constructed from the design matrix independently of the outcome in PCR, PLS uses the outcome variable to guide the construction of these latent variables, which makes it a supervised method and might offer better performance. PLS is a suitable analytical tool for data that are high-dimensional, strongly collinear and noisy [12]. While training, I set the number of components from 1 to 100. Performance decreases after using the first few components and remains stable after so I did not tune over a larger range.

Support Vector Classifier: SVC constructs a hyperplane in the p-dimensional feature space that separates the two classes. This hyperplane is a (p-1)-dimensional subspace and depends on the support vectors (datapoints that are on the margins on its two sides). The goal is to construct a hyperplane such that the distances from the support vectors are maximized. With the linear kernel, we have the cost hyperparameter C [9]. As C decreases, less datapoints are allowed to be misclassified, which leads to a narrower margin. This also implies lower bias and higher variance. Here, I experimented and saw SVC and SVM radial kernel have similar performance, which indicates no need to find a hyperplane in the expanded feature space. Here, I tuned the model over the grid $[e^{-10}, e^{-4}]$ of equally spaced values.

Random Forest: Since decision trees suffer from the curse of dimensionality and tend to overfit, ensemble methods such as bagging solves this by growing trees on bootstrap samples and aggregate votes across many trees to reduce variability. Random forest further builds on the idea of bagging by randomly select a subset of variables to grow the trees as well as prune the trees by controlling tree depth or node size. This random feature subset component helps decorrelate the trees in case there are some features that are dominant and appear in the majority of trees under bagging. Thanks to decorrelation, the variance can be further reduced. To optimize the number of randomly selected predictors m, I set the grid to be between 1 and 30, although people tend to set $m = \sqrt{p}$ for classification trees. For RF model fitted on unreduced data, I chose a wider grid (12 to 60) for m.

Gradient Boosting Machine (GBM) can be seen as an extension of bagging, in which the trees are grown sequentially and using the information from the trees grown previously. It adds weak learners (shrunken trees with few terminal nodes) to the current model to update the residuals. This guides how new weak learners can be added to the current fitted function to improve in areas where it does not do well. Overall, this can be seen as a learning process, and slow learning (more shrunken added weak learner) is preferable since it prevents the boosted tree from overlearning, which may cause high variance. However, this can be computationally expensive, as more trees also need growing for more stability. **XGBoost** improves on GBM by using weighted quantile sketch procedure, which only tests quantiles to approximate where to split instead of all possible points. This helps the exact greedy algorithm be more efficient. A cache-aware prefetching algorithm is also introduced to reduce the overhead computation cost while optimizing split (by storing gradients and Hessians in cache memory for faster access) [3].

For XGBoost, after some experimenting, the learning rate eta was set to 0.05, max tree depth is optimized over grid from 5 to 15, min_child_weight = 1, colsample_bytree is optimized over [0.1,0.15,0.2], and number of boostes trees is set to 100.

Results

Generally, we can see that applying screeners help boost performance for most learners. In terms of Precision-Recall AUC for different pipelines (**Figures 4, 5, 7, 9**), we observe comparably good performances across pipelines and learners (most are above 0.85), especially for PLS-SVC and PLS-RF. When looking at ROC AUC curves in **Figures 4, 6, 8, 10**, we see more distinct improvements across pipelines. PLS-SVC, PLS-RF, PLS-XGBoost all have the highest CV ROC AUC. Interestingly, embedded feature selection methods like PLS and Elastic net, as multivariate filters, returns more useful features, even for tree-based methods like RF and XGBoost, compared to PCA. The same result is observed for Support Vector Classifier (Enet-SVC, PLS-SVC, and Limma-PLS-SVC have good performance compared to SVC in the unreduced feature space). **Figure 11** shows PLS-SVC, PLS-XGBoost, and PLS-RF have the highest mean ROC AUC across resamples.

SVC fitted on 17 metabolites selected from Enet also gives relatively good performance while also being the most parsimonious classifier among the models considered. Additionally, SVC optimization problem can be rephrased to be similar to a logistic regression with a ridge penalty problem (the smoother binomial loss function can approximate the hinge loss) [8, 9] This connection between linear SVM and penalized regression potentially allows more meaningful variables being selected for SVC from Enet. Overall, PC-based learners do not seem to have great classification performance.

In terms of computation time, **Table 2** confirms the hypothesis that screening variables helps decrease computation time while preserving performance, if not enhancing it in some cases. As expected, applying learners on full datasets result in very long computation time and potentially lackluster performance. Computation expense decreases significantly with a reduced feature space. Overall, using only univariate filtering seems to be less time-saving than adding multivariate filtering, since the feature space can remain large.

I then explored the effect of two metabolites among the “most important”, “meta1076” and “meta335”, using Partial Dependence Plots (PDP) (**Figures 14, 15**) from SVC and XGBoost models. PDP show the marginal effect of certain metabolites on the probability of incident AD. The plots suggest higher levels of these metabolites are associated with decreased risk of incident AD. In addition, the effects in SVC are linear, which is highly interpretable but might oversimplify the true relationship. Meanwhile, the effects are non-linear for XGBoost and thus can offer finer details. They are also not too squiggly or varying dramatically, which indicates little overfitting.

Discussion

Widely used learners for classification problems include regularized regression, support vector machine, ensemble tree-based methods, or variations of decision tree boosting. Among these, regularized regression is relatively fast, more interpretable, and easy to implement. However, these were built upon OLS with added penalties, so they model the relationship linearly, which can, in most cases, oversimplify reality. Ensemble methods and boosting might be more attractive choices, but when p>>n, the algorithm requires more computational power, and an exhaustive search for good combinations of features might not be feasible.

Since our untargeted metabolomics data gives rise to a large input space with high amount of noise, fitting a model with a random component like random forest might suffer from utilizing irrelevant variables and/or overfitting in other cases. Therefore, fast and efficient screening pipelines might be helpful in reducing the dimension of the feature space to only relatively meaningful variables [11], which was observed to increase both predictive performance and reduce computation time in this analysis.

Despite being fast and seemingly efficient in picking informative features, methods like PLS or Enet are built-in/embedded feature selection. This means the model will only select predictors for a specific optimization problem defined by loss function at hand. Therefore, the representation of the data chosen by the PLS or Enet might not be the best for the purpose of prediction using methods like RF and XGBoost. In the end, one needs to be careful as using separate learners with embedded variable selection to screen variables might not guarantee good properties, and we risk losing important variables conducive to better performance. However, this approach is still employed in some analyses [5, 6, 12].

Since lab-based enrichment analysis needs to be conducted, the metabolites in this data do not have labels yet. However, we see overlaps in the most important variables selected by different algorithms, which might suggest some underlying biological connection. PDP offered some perspective on how these metabolites may affect the probability of incident AD.

Limitations: A potential drawback of univariate filtering is that it can eliminate variables that are useful when considered in higher-dimensional space with other variables but are meaningless on their own. This might be the case in this analysis, where learners' performance using features selected by just the multivariate filter is generally better than using both univariate and multivariate filter. Additionally, wrapper methods like recursive feature elimination was not applied due to time constraint. This might change the results. For example, when constructing a hyperplane in SVC, a smaller subset of features might induce a better separating hyperplane than the full set of features [2].

Next, since I evaluated screening methods on a real dataset, I did not have information about the true features that are relevant with respect to the outcome, so I can only look at prediction performance. Lastly, not having a hold-out set to test classification performances means the observed cross-validated AUC could have been overly optimistic.

Future Directions: Simulation studies is likely better to evaluate the effectiveness of different variable selection methods and pipelines under different characteristics of the data. Wrapper methods can be considered after the feature space has been reduced to further improve performance while keeping computation cost relatively low.

Other unsupervised method like clustering or graphical models might also be helpful in understanding the bodily functions and the relationships/pathways between these metabolites, once they have been labeled.

References

[1] Alzheimer's Association (2020). “What is Alzheimer's Disease?” https://www.alz.org/alzheimers-dementia/what-is-alzheimers

[2] Atla, A., et al. (2011). “Sensitivity of different machine learning algorithms to noise”. CCSC: Mid-South Conference [https://dl.acm.org/doi/pdf/10.5555/1961574.1961594]

[3] Chen, T., & Guestrin, C. (2016) “XGBoost: A scalable tree boosting system.” **arXiv** https://arxiv.org/abs/1603.02754

[4] Determan Jr., CE. (2015). “Optimal Algorithm for Metabolomics Classification and Feature Selection varies by Dataset.” **International Journal of Biology**; 7(1); doi: 10.5539/ijb.v7n1p100

[5] Dhouha, G., Melanie, P., Marion, B., et al. (2016) “Feature Selection Methods for Early Predictive Biomarker Discovery Using Untargeted Metabolomic Data” Frontiers in Molecular Biosciences. 3(30). DOI=10.3389/fmolb.2016.00030

[6] Ghaffari, MH., Jahanbekam, A., Sadri, H., Schuh, K. Koch, C., Sauerwein, H., et al. (2019) “Metabolomics meets machine learning: Longitudinal metabolite profiling in serum of normal versus overconditioned cows and pathway analysis” **Journal of Dairy Science** 102(2): 11561-11585. https://doi.org/10.3168/jds.2019-17114.

[7] Huan Xu, Constantine Caramanis (2012) “Sparse Algorithms are not Stable: A No-free-lunch Theorem”. **IEEE Trans Pattern Anal Mach Intell** 34(1):187-93

[8] Jaggi M., (2014). “An Equivalence between the Lasso and Support Vector Machines” **arXiv**; 1303.1152v2 https://arxiv.org/pdf/1303.1152.pdf

[9] James, G., Witten, D., Hastie, T.„ Tibshirani, R. (2013). An Introduction to Statistical Learning: with Applications in R. Springer.

[10] Mehmood, T, Saebo, S, Liland, K. (2020) Comparison of variable selection methods in Partial Least Squares Regression. **Journal of Chemometrics**. 34:e3226. https://doi.org/10.1002/cem.3226

[11] Perez-Riverol Y, Kuhn M, Vizcaino JA, Hitz MP, Audain E. (2017). Accurate and fast feature selection workflow for high-dimensional omics data. **PLoS One** 12(12):e0189875. doi:10.1371/journal.pone.0189875

[12] Piles, M., Fernandez-Lozano, C., Velasco-Galilea, M. et al. (2019) Machine learning applied to transcriptomic data to identify genes associated with feed efficiency in pigs. **Genet Sel Evol** 51(10). https://doi.org/10.1186/s12711-019-0453-y

[13] Saito, T., Rehmsmeier, M. (2015). “The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets.” **PLOS one**. https://doi.org/10.1371/journal.pone.0118432

[14] Shigemizu, D., Akiyama, S., Higaki, S. et al. (2020). "Prognosis prediction model for conversion from mild cognitive impairment to Alzheimer's disease created by integrative

analysis of multi-omics data. **Alz Res Therapy** 12(145). https://doi.org/10.1186/s13195-020-00716-0

[15] Smyth GK. (2004). Linear models and empirical bayes methods for assessing differential expression in microarray experiments. **Stat Appl Genet Mol Biol**; 3:Article3; doi: 10.2202/1544-6115.1027

Appendix

Table 1: Descriptive Statistics of Baseline Covariates across Levels of Outcome

	No AD (N=60)	AD (N=183)	Total (N=243)
Sex			
- Male	39 (65.0%)	111 (60.7%)	150 (61.7%)
- Female	21 (35.0%)	72 (39.3%)	93 (38.3%)
APOE4 presence			
- 0	60 (100.0%)	183 (100.0%)	243 (100.0%)
Race/Ethnicity			
- Whites	56 (93.3%)	170 (92.9%)	226 (93.0%)
- Non-whites	4 (6.7%)	13 (7.1%)	17 (7.0%)
Age at blood draw visit			
- Mean (SD)	54.52 (5.29)	50.51 (6.57)	51.50 (6.50)
- Median (Q1 - Q3)	54.41 (50.92, 57.69)	50.04 (47.39, 53.74)	51.28 (47.94, 55.02)

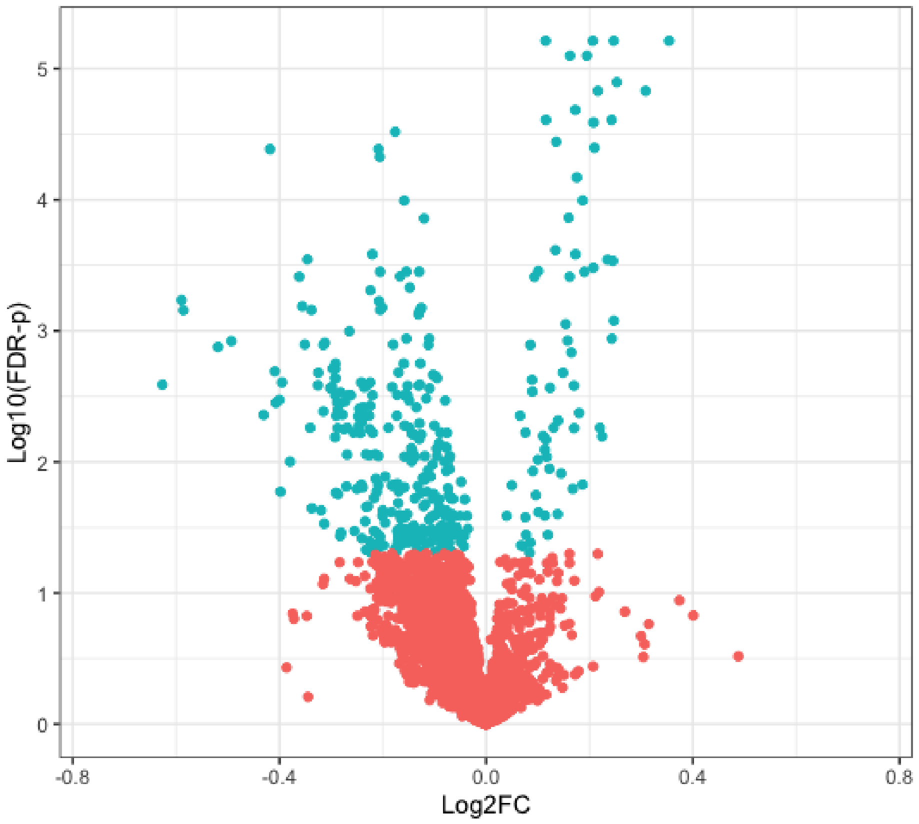


Figure 1: Volcano Plot (DE Analysis)

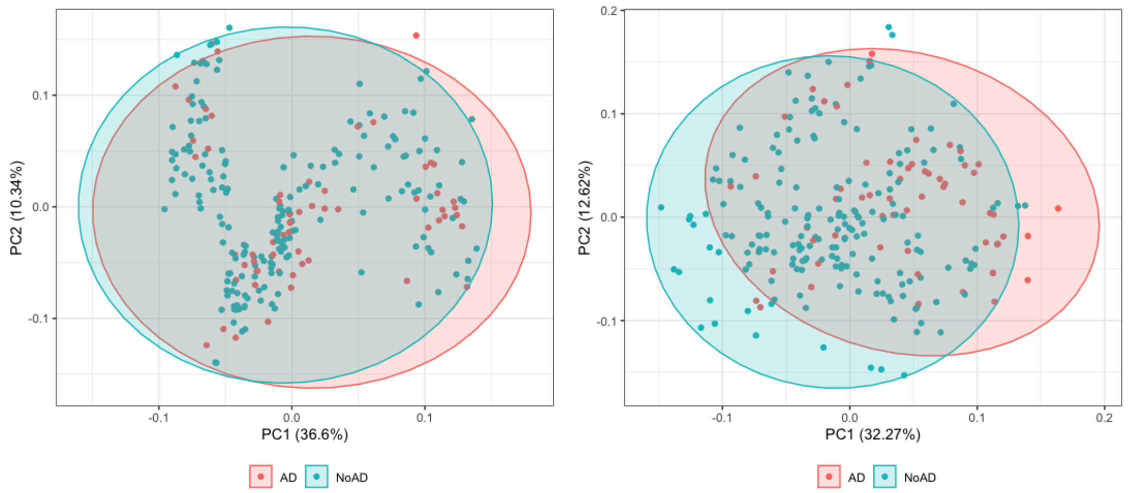


Figure 2: PCA - first 2PCs from full data (left) vs. Limma-PCA (right)

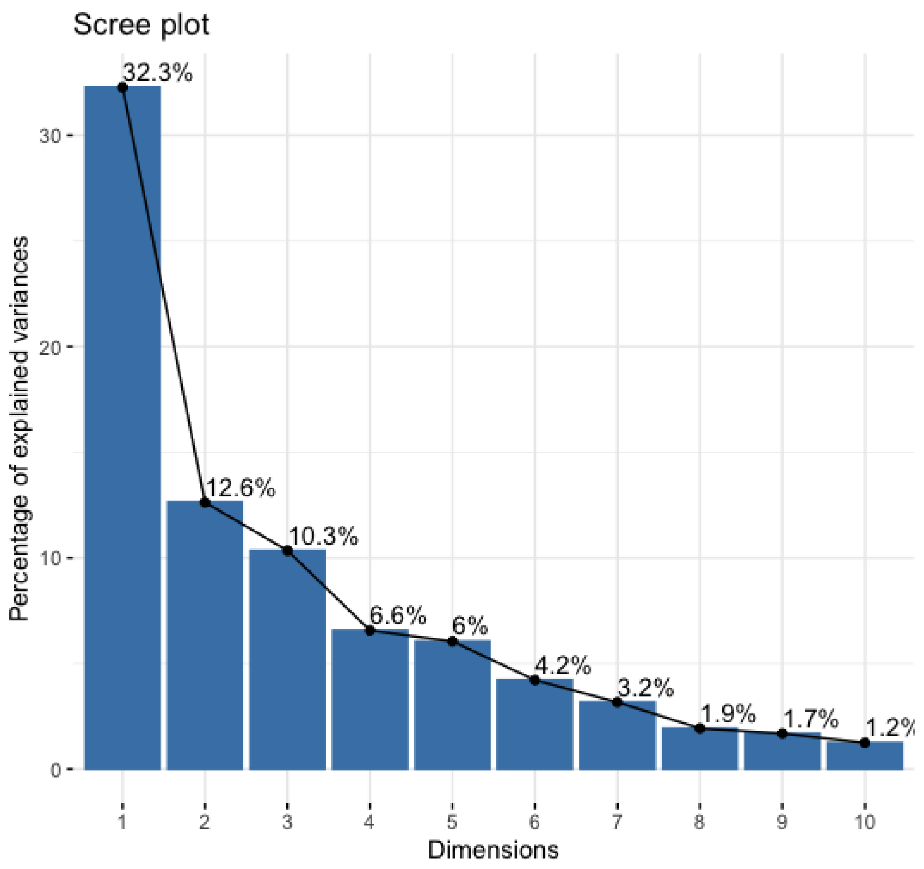


Figure 3: Scree plot for PCA on 366 metabolites from DE analysis

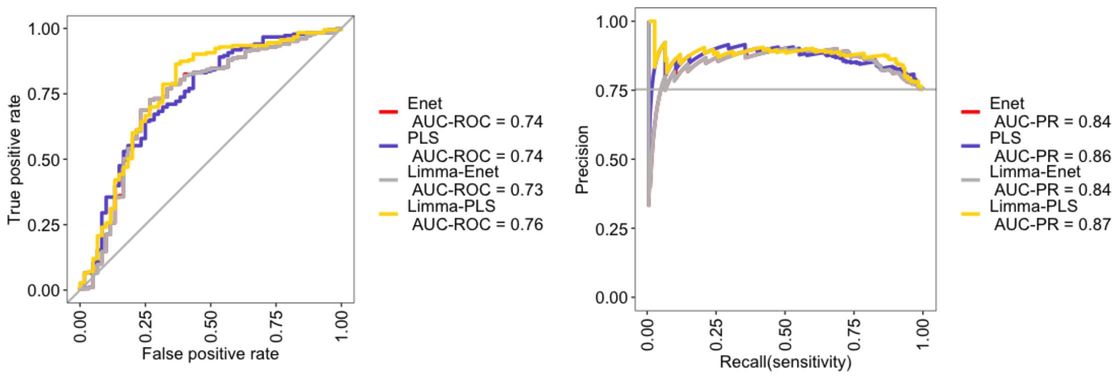


Figure 4: ROC AUC and PR AUC for PLS and Enet

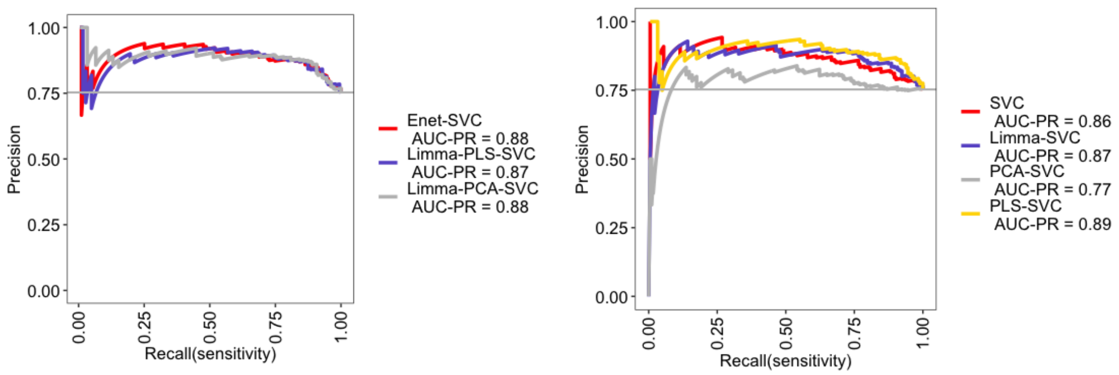


Figure 5: PR AUC for SVC

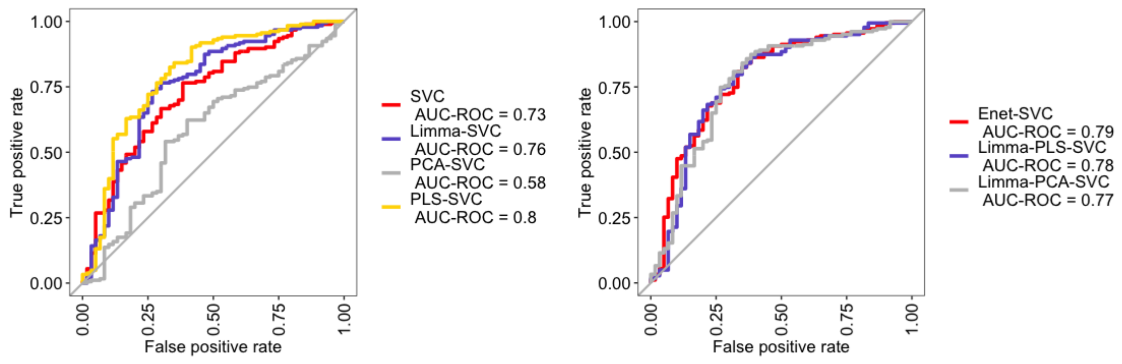


Figure 6: ROC AUC for SVC

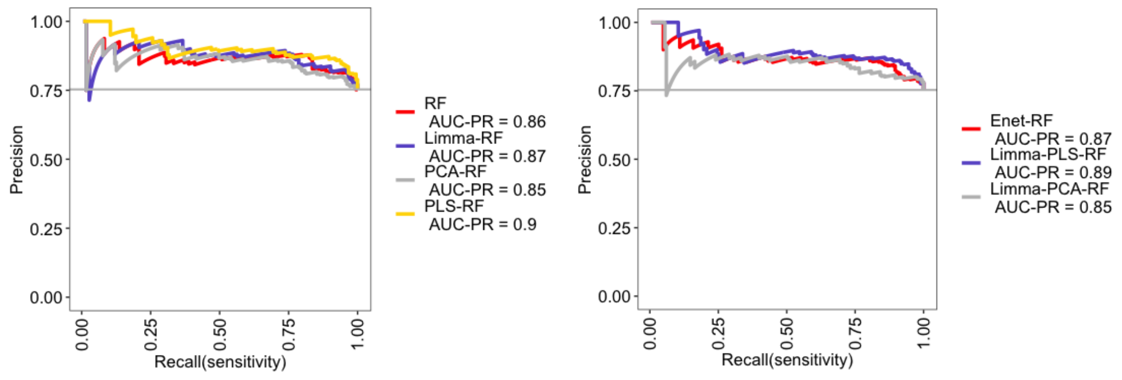
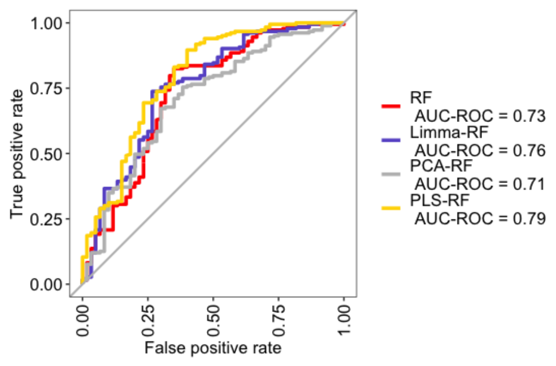


Figure 7: PR AUC for Random Forest



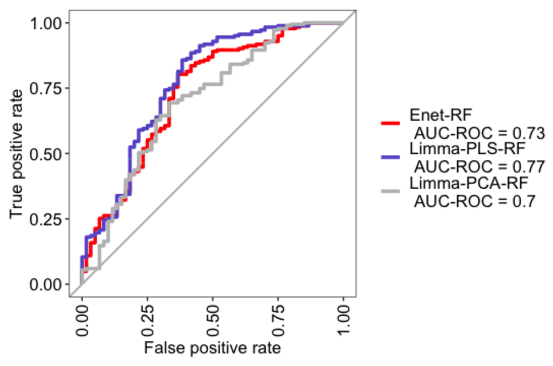
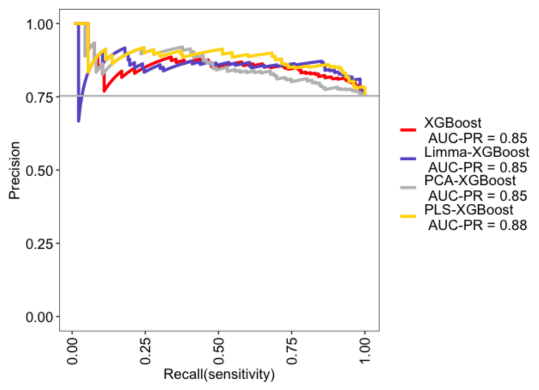


Figure 8: ROC AUC for Random Forest



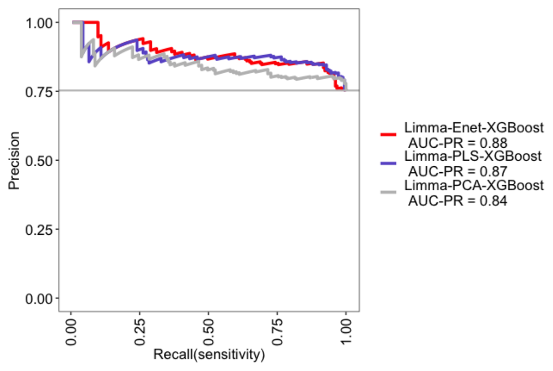
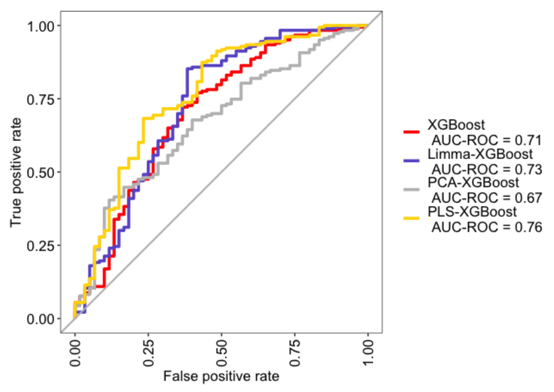


Figure 9: PR AUC for XGBoost



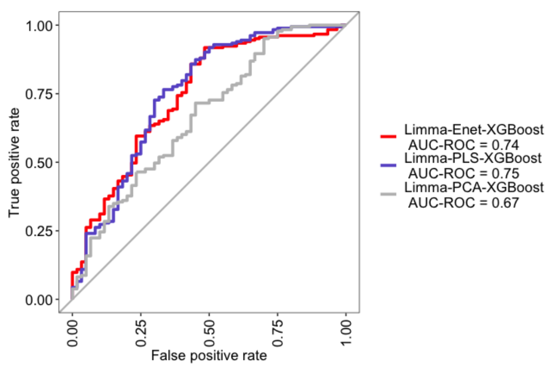


Figure 10: ROC AUC for XGBoost

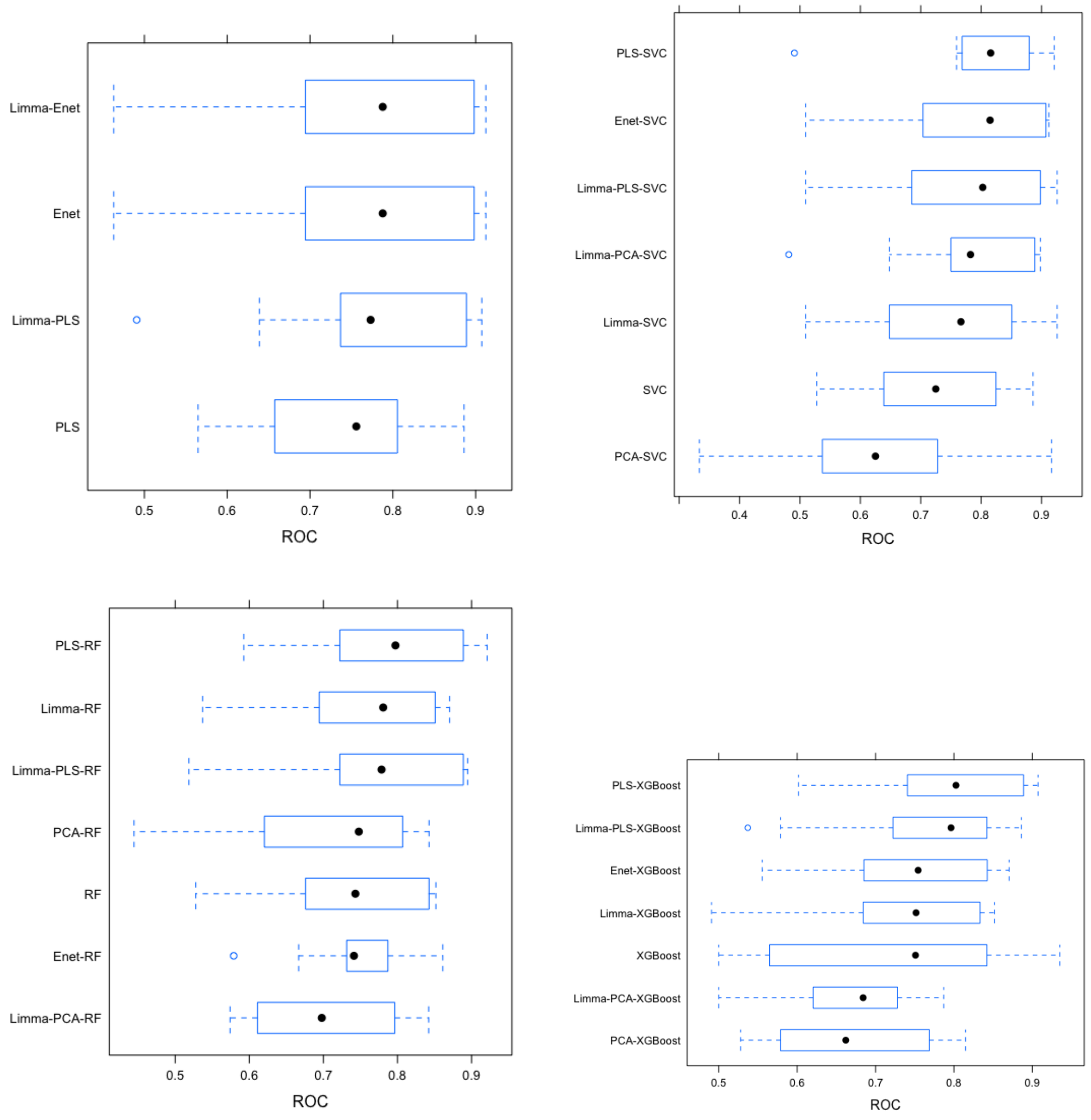
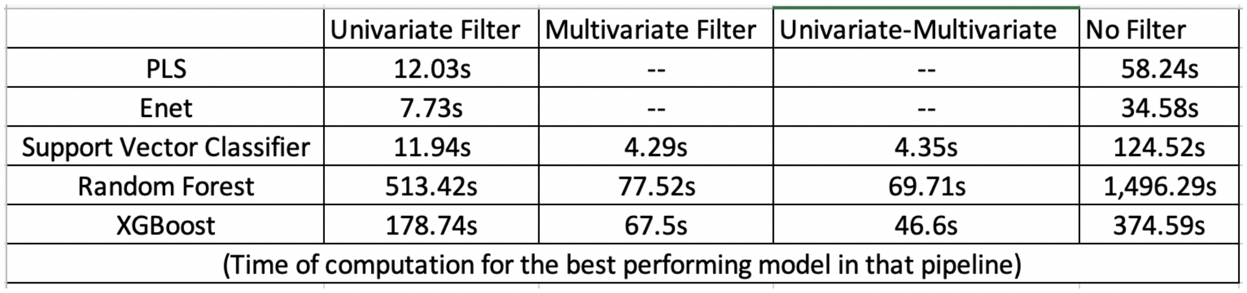


Figure 11: ROC AUC for all models from resamples



	Univariate Filter	Multivariate Filter	Univariate-Multivariate	No Filter
PLS	12.03s	--	--	58.24s
Enet	7.73s	--	--	34.58s
Support Vector Classifier	11.94s	4.29s	4.35s	124.52s
Random Forest	513.42s	77.52s	69.71s	1,496.29s
XGBoost	178.74s	67.5s	46.6s	374.59s
(Time of computation for the best performing model in that pipeline)				

Figure 12: Table 2: Elapsed time for each pipeline

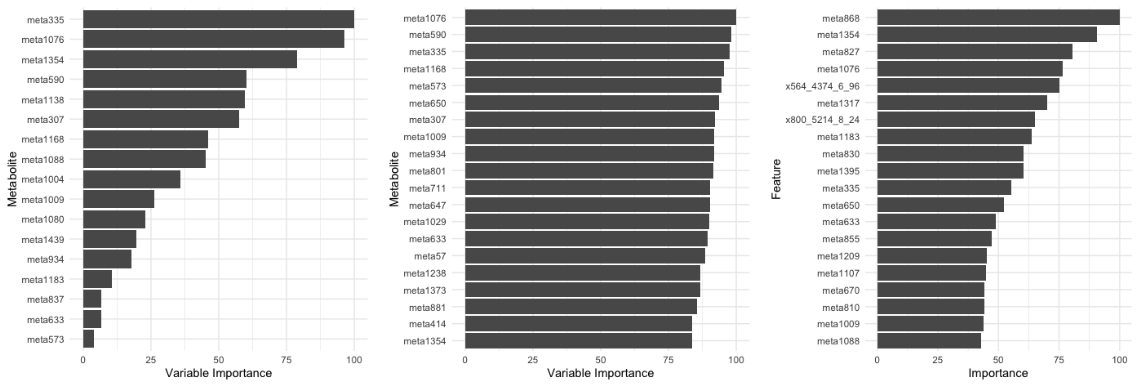


Figure 13: Variable Importance Plots (left-right: Enet - SVC - XGB)

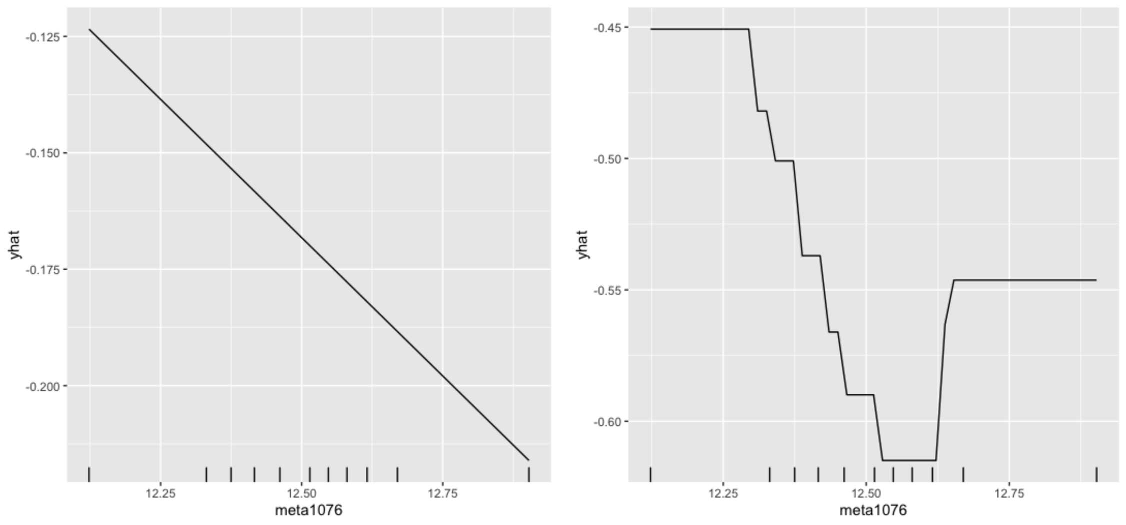


Figure 14: Partial Dependence Plots for meta1076 (SVC - XGBoost)

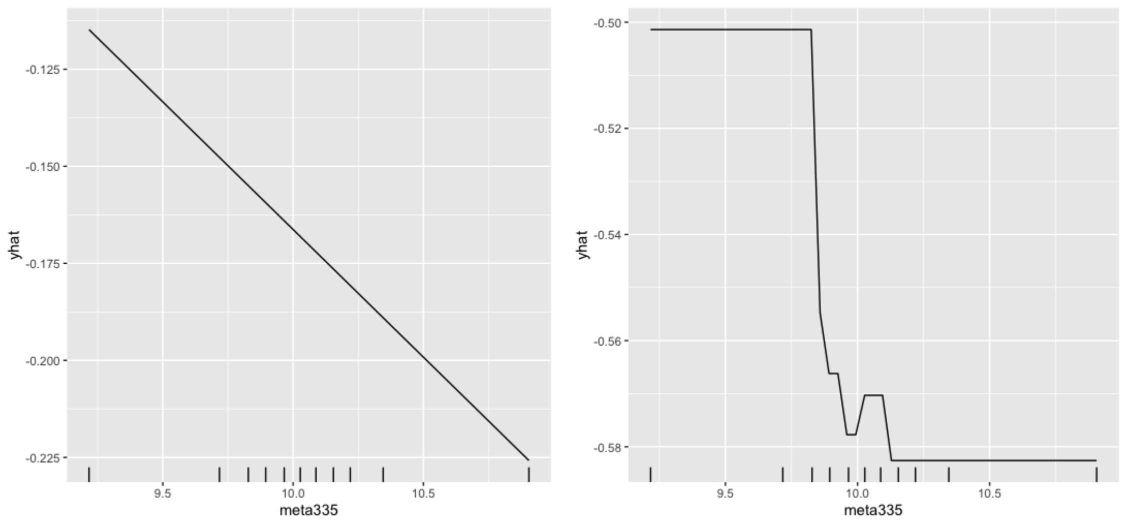


Figure 15: Partial Dependence Plots for meta335 (SVC - XGBoost)

Codes

```r
#import data
data_ad = read_csv("data_ad_incident.csv")
data_phen = data_ad %>% dplyr::select(overall_ad, sex, age_blood_draw, apo_e_geno, ethni
  mutate(apo_e_geno = ifelse(apo_e_geno == "3/4" | apo_e_geno == "4/4", 1, 0),
         apo_e_geno = ifelse(is.na(apo_e_geno) == TRUE & overall_ad == "AD", 1,
                             ifelse(is.na(apo_e_geno) == TRUE & overall_ad == "NoAD", 0

data_met = data_ad %>% dplyr::select(-par_id, -overall_ad, -sex, -age_blood_draw, -apo_
#log transform
vars <- c(6:2713); data_phen[,vars] <- lapply(data_phen[,vars] + 0.001, log)
#descriptive statistics
my_controls <- tableby.control(total = T, test = F, numeric.stats = c("meansd", "median

data_tab1 =  data_ad %>% dplyr::select(overall_ad, sex, ethnicity, age_blood_draw, apo_e
  mutate(overall_ad = factor(overall_ad, labels = c("No AD", "AD")), sex = factor(sex,
         ethnicity = factor(ethnicity, labels = c("Whites", "Non-whites")), apo_e_geno =

my_labels <- list(overall_ad = "Incident AD (all visits)", sex = "Sex", apo_e_geno = "AP

table <- tableby(overall_ad~sex + apo_e_geno + ethnicity + age_blood_draw , data=data_ta
summary(table, title = "Descriptive Statistics of Baseline Covariates across Levels of (

#Univariate filtering
data_tf = t(log(data_met+0.001))
#linear model using limma
design <- model.matrix(~overall_ad + age_blood_draw + as.factor(sex) + ethnicity + apo_
colnames(design) <- c("Intercept", "ad", "age", "sex", "apoe", "ethnic")
fit <- lmFit(data_tf, design)
fit <- eBayes(fit)
results <- decideTests(fit[, 2])
summary(results)
stats <- topTable(fit, coef = "ad", number = nrow(fit), sort.by = "P")
hist(stats[, "adj.P.Val"], xlab = "q-values", main = "Limma linear model")
met.limma.lm = stats %>% filter(adj.P.Val < 0.05)
met.limma.lm = rownames(met.limma.lm)
stats.df = as_tibble(stats) %>%
  mutate(metabolite = rownames(stats),
         logp = abs(log(adj.P.Val, 10)),
         sig = adj.P.Val <= 0.05)
#volcano plot
ggplot(stats.df, aes(x = logFC, y = logp, col = sig)) + geom_point() +
  scale_x_continuous(name="Log2FC", limits=c(-0.75, 0.75)) +labs(y = "Log10(FDR-p)") +
  theme_bw() + theme(legend.position = "none")
```

```
#Data from univariate filtering step
data.ad.lm = dplyr::select(data_phen, overall_ad, all_of(met.limma.lm))

#Multivariate Filtering
#PCA
pca.phen = prcomp(data_phen[,6:2013], scale = T) #full data
autoplot(pca.phen, data = data_phen[,c(1,6:2013)], colour = "overall_ad",
         frame = TRUE, frame.type = "norm") + theme_bw() + theme(legend.position = "bot
summary(pca.phen)
fviz_eig(pca.phen, addlabels = TRUE)

#pca from limma selected variables
pca.lm = prcomp(data.ad.lm[,-1], scale = T)
autoplot(pca.lm, data = data.ad.lm, colour = "overall_ad",
         frame = TRUE, frame.type = "norm") + theme_bw() + theme(legend.position = "bot
summary(pca.lm)#about 30PCs give 90% variance explained
fviz_eig(pca.lm, addlabels = TRUE)

# Data from PCA
data.full.pca.s = as.data.frame(pca.phen$x[,1:4]) %>% mutate(overall_ad = data_phen$ove
data.lm.pca.s = as.data.frame(pca.lm$x[,1:4]) %>% mutate(overall_ad = data_phen$overall
data.full.pca.l = as.data.frame(pca.phen$x[,1:30]) %>% mutate(overall_ad = data_phen$ov
data.lm.pca.l = as.data.frame(pca.lm$x[,1:30]) %>% mutate(overall_ad = data_phen$overal

#PLS
#lm-pls
ctrl.pls <- trainControl(method = "cv", number = 10, sampling = "up",
                         summaryFunction = twoClassSummary, classProbs = TRUE, savePredictio
set.seed(2020)
system.time({
lm.pls <- train(overall_ad~., data.ad.lm, metric = "ROC",
  method = "pls", preProc = c("zv", "center", "scale"),
  tuneLength = 100, trControl = ctrl.pls)}) #time 12.031s
median(lm.pls$resample$ROC) #0.77315
mean(lm.pls$resample$ROC) #0.7678
sd(lm.pls$resample$ROC) #0.1287
lm.pls.metab = varImp(lm.pls)$importance %>% arrange(desc(Overall))
lm.pls.metab = rownames(lm.pls.metab)[1:100]

#pls
set.seed(2020)
system.time({
full.pls <- train(overall_ad~., data_phen[,c(1,6:2713)], metric = "ROC",
  method = "pls", preProc = c("zv", "center", "scale"),
  tuneLength = 100, trControl = ctrl.pls)
}) #time 58.24s
median(full.pls$resample$ROC) #0.7558
mean(full.pls$resample$ROC) #0.7428
sd(full.pls$resample$ROC) #0.106
full.pls.metab = varImp(full.pls)$importance %>% arrange(desc(Overall))
full.pls.metab = rownames(full.pls.metab)[1:100]

data.lm.pls = dplyr::select(data_phen, overall_ad, all_of(lm.pls.metab)) #data from PLS
data.pls = dplyr::select(data_phen, overall_ad, all_of(full.pls.metab))

##Elast net tuning grids
tuneGrid=expand.grid(alpha = c(0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 0.8), lambda=2^(seq(-4,1
trainControl <- trainControl(method = "cv", number = 10, sampling = "up",
                             summaryFunction = twoClassSummary, classProbs = TRUE, savePredict
set.seed(2020) #lm-Enet
system.time({
enet.lm <- train(overall_ad ~. , data.ad.lm,
                 method = "glmnet", trControl = trainControl,
                 tuneGrid = tuneGrid, parallelize = T,
                 preProc = c("center", "scale"), metric = "ROC", family="binomial")}) #
median(enet.lm$resample$ROC) #0.7878
mean(enet.lm$resample$ROC) #0.7581
sd(enet.lm$resample$ROC) #0.1436
enet.lm.coeff = coef(enet.lm$finalModel, enet.lm$bestTune$lambda)
enet.lm.coeff = data.frame(name = enet.lm.coeff@Dimnames[[1]][enet.lm.coeff@i + 1], coe
#17 metabolites selected

#full data
set.seed(2020)
system.time({
enet.full <- train(overall_ad ~. , data_phen[,c(1,6:2713)],
                   method = "glmnet", trControl = trainControl,
                   tuneGrid = tuneGrid, parallelize = T,
                   preProc = c("center", "scale"), metric = "ROC", family="binomial")})
median(enet.full$resample$ROC) #0.7878
mean(enet.full$resample$ROC) #0.7581
sd(enet.full$resample$ROC) #0.1436
enet.full.coeff = coef(enet.full$finalModel, enet.full$bestTune$lambda)
enet.full.coeff = data.frame(name = enet.full.coeff@Dimnames[[1]][enet.full.coeff@i + 1
#same 17 metabolites selected - CV AUC about 0.76

evalm(list(enet.full, full.pls, enet.lm, lm.pls),gnames=c('Enet', 'PLS', 'Limma-Enet',
resamp.pls.enet = resamples(list("Enet" = enet.full, "Limma-PLS" = lm.pls, "PLS" = full
```

```r
bwplot(resamp.pls.enet, metric = "ROC")

##Create design matrix using model matrix before ML algorithm
#limma-pls
X.lm.pls = model.matrix(overall_ad ~., data = data.lm.pls)[,-1]
y.lm.pls = data.lm.pls$overall_ad

#pls
ad.noad.pls = data_phen %>% dplyr::select(overall_ad, one_of(full.pls.metab))
X.pls = model.matrix(overall_ad ~., data = ad.noad.pls)[,-1]
y.pls = ad.noad.pls$overall_ad

#lm-enet
data.lm.enet = data_phen %>% dplyr::select(overall_ad, one_of(enet.lm.coeff))
X.lm.enet = model.matrix(overall_ad ~., data = data.lm.enet)[,-1]
y.lm.enet = data.lm.enet$overall_ad

#SVC
ctrl <-trainControl(method = "cv", number = 10, summaryFunction = twoClassSummary,
                    classProbs = TRUE, sampling = "up", savePredictions = T)

svmr.grid <-expand.grid(C =exp(seq(-10,-4,len=10)),
                        sigma =exp(seq(-8,1,len=10)))
#lm-svc
set.seed(2020)
system.time({
svml.lm <-train(overall_ad ~., data.ad.lm, method = "svmLinear2", allowParallel = TRUE,
                preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                tuneGrid =data.frame(cost =exp(seq(-10,-4,len=10))))
}) #time 11.94s
median(svml.lm$resample$ROC) #0.7665
mean(svml.lm$resample$ROC) #0.7523
sd(svml.lm$resample$ROC) #0.1325

#lm-enet-svc
set.seed(2020)
system.time({
svml.lm.enet <-train(X.lm.enet, y.lm.enet, method = "svmLinear2", allowParallel = TRUE,
                preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                tuneGrid =data.frame(cost =exp(seq(-10,-4,len=10))))}) #time 1.96s
median(svml.lm.enet$resample$ROC) #0.815
mean(svml.lm.enet$resample$ROC) #0.785
sd(svml.lm.enet$resample$ROC) #0.1265
```

```
#SVM radial kernel
set.seed(2020)
system.time({
svmr.fit <-train(overall_ad ~., data.lm.pca.s,
                 method = "svmRadial", allowParallel=TRUE, metric = "ROC",
                 preProc = c("scale", "center"), tuneGrid = svmr.grid, trControl = ctrl)
median(svmr.fit$resample$ROC) #0.815
mean(svmr.fit$resample$ROC) #0.788
sd(svmr.fit$resample$ROC) #0.112

#pca-svc
set.seed(2020)
system.time({
svml.pca <-train(overall_ad ~., data.full.pca.s, method = "svmLinear2", allowParallel =
                 preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                 tuneGrid =data.frame(cost =exp(seq(-10,-3,len=10))))}) #time 1.75s
median(svml.pca$resample$ROC) #0.625
mean(svml.pca$resample$ROC) #0.634
sd(svml.pca$resample$ROC) #0.157

#pls-svm
set.seed(2020)
system.time({
svml.pls <-train(X.pls, y.pls, method = "svmLinear", allowParallel = TRUE,
                 preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                 tuneGrid =data.frame(C =exp(seq(-10,-3,len=10))))}) #time 4.29s
median(svml.pls$resample$ROC) #0.8157
mean(svml.pls$resample$ROC) #0.7987
sd(svml.pls$resample$ROC) #0.121

#lm-pca-svm
set.seed(2020)
system.time({
svml.lm.pca <-train(overall_ad ~., data.lm.pca.s,method = "svmLinear2", allowParallel =
                 preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                 tuneGrid =data.frame(cost =exp(seq(-10,-3,len=10))))}) #time 1.7s
median(svml.lm.pca$resample$ROC) #0.7824
mean(svml.lm.pca$resample$ROC) #0.7714
sd(svml.lm.pca$resample$ROC) #0.1289

#lm-pls-svm
set.seed(2020)
system.time({
svml.lm.pls <-train(X.lm.pls, y.lm.pls, method = "svmLinear2", allowParallel = TRUE,
```

```r
                    preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                    tuneGrid =data.frame(cost =exp(seq(-10,-4,len=10))))
}) #time 4.35s
median(svml.lm.pls$resample$ROC) #0.8026
mean(svml.lm.pls$resample$ROC) #0.785
sd(svml.lm.pls$resample$ROC) #0.129

#full SVM
set.seed(2020)
system.time({
svml.full <-train(overall_ad ~., data_phen[,c(1,6:2713)], method = "svmLinear2", allowPa
                    preProc = c("scale", "center"), metric = "ROC", trControl = ctrl,
                    tuneGrid =data.frame(cost =exp(seq(-10,-3,len=10))))}) #time 124s
median(svml.full$resample$ROC) #0.707
mean(svml.full$resample$ROC) #0.7165
sd(svml.full$resample$ROC) #0.152

#AUC
evalm(list(svml.lm.enet, svml.lm.pls, svml.lm.pca),gnames=c('Enet-SVC', 'Limma-PLS-SVC'
evalm(list(svml.full, svml.lm, svml.pca, svml.pls),gnames=c('SVC', 'Limma-SVC', 'PCA-SV
#resamples
resamp.svm = resamples(list("Limma-SVC" = svml.lm, "Limma-PCA-SVC" = svml.lm.pca, "Limma
bwplot(resamp.svm, metric = "ROC")

#Random forest
ctrl_rf = trainControl(method = "cv", number = 10, summaryFunction = twoClassSummary,
                        sampling = "up", classProbs = TRUE, savePredictions = T)
doMC::registerDoMC(6)
rf.grid = expand.grid(mtry = c(seq(1,10, by = 1)), splitrule = "gini", min.node.size = 

#lm-rf
set.seed(2020)
rf.grid.lm = expand.grid(mtry = c(seq(2,22, by = 1)), splitrule = "gini", min.node.size
system.time({
rf.lm <- train(overall_ad~., data.ad.lm, method = "ranger", tuneGrid = rf.grid.lm,
                  metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #tim
median(rf.lm$resample$ROC) #0.7807
mean(rf.lm$resample$ROC) #0.7597
sd(rf.lm$resample$ROC) #0.1087

#pls-rf
set.seed(2020)
system.time({
rf.pls <- train(overall_ad~., data.pls, method = "ranger", tuneGrid = rf.grid,
                metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #77.
median(rf.pls$resample$ROC) #0.7972
mean(rf.pls$resample$ROC) #0.7884
sd(rf.pls$resample$ROC) #0.106

#pca-rf
set.seed(2020)
system.time({
rf.pca <- train(overall_ad~., data.full.pca.l, method = "ranger", tuneGrid = rf.grid,
                metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #40.
median(rf.pca$resample$ROC) #0.748
mean(rf.pca$resample$ROC) #0.7117
sd(rf.pca$resample$ROC) #0.123

#lm-pca-rf
set.seed(2020)
system.time({
rf.lm.pca <- train(overall_ad~., data.lm.pca.l, method = "ranger", tuneGrid = rf.grid,
                metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #72.
median(rf.lm.pca$resample$ROC) #0.7746
mean(rf.lm.pca$resample$ROC) #0.75102
sd(rf.lm.pca$resample$ROC) #0.12

#lm-pls-rf
rf.grid = expand.grid(mtry = c(seq(1,10, by = 1)), splitrule = "gini", min.node.size = 
set.seed(2020)
system.time({
rf.lm.pls <- train(overall_ad~., data.lm.pls, method = "ranger", tuneGrid = rf.grid, pre
                metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #69.
median(rf.lm.pls$resample$ROC) #0.77412
mean(rf.lm.pls$resample$ROC) #0.77
sd(rf.lm.pls$resample$ROC) #0.1186

#lm-enet-rf
system.time({
rf.lm.enet <- train(X.lm.enet, y.lm.enet, method = "ranger", tuneGrid = rf.grid, preProc
                metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #30.
median(rf.lm.enet$resample$ROC) #0.816
mean(rf.lm.enet$resample$ROC) #0.8112
sd(rf.lm.enet$resample$ROC) #0.103

#rf
set.seed(2020)
rf.grid.full = expand.grid(mtry = c(seq(12,60, by = 3)), splitrule = "gini", min.node.s
system.time({
rf.full <- train(overall_ad~., data_phen[,c(1,6:2713)], method = "ranger", tuneGrid = rf
                metric = "ROC", importance = "permutation", trControl = ctrl_rf)}) #1496
ggplot(rf.full, highlight = TRUE)
median(rf.full$resample$ROC) #0.7432
mean(rf.full$resample$ROC) #0.7372
sd(rf.full$resample$ROC) #0.1042

#resamples
resamp.rf = resamples(list("Enet-RF" = rf.lm.enet, "Limma-PCA-RF" = rf.lm.pca, "PCA-RF"
bwplot(resamp.rf, metric = "ROC")

#ROC and PR curves
evalm(list(rf.lm.enet, rf.lm.pls, rf.lm.pca),gnames=c('Enet-RF', 'Limma-PLS-RF', 'Limma-
evalm(list(rf.full, rf.lm, rf.pca, rf.pls),gnames=c('RF', 'Limma-RF', 'PCA-RF', 'PLS-RF

#XGBoost
ctrl_xgb <- trainControl(method = "cv", number = 10, sampling = "up", classProbs = TRUE
                         summaryFunction = twoClassSummary, allowParallel = TRUE, savePredic

parametersGrid <-  expand.grid(eta = 0.05, colsample_bytree=c(0.1,0.15,0.2), max_depth=c
                               nrounds=100, subsample = 0.5, gamma=1, min_child_weight=2

#lm-xgb
set.seed(2020)
system.time({
xgb.lm <- train(overall_ad~., data.ad.lm, method = "xgbTree", metric = "ROC",
                trControl = ctrl_xgb, tuneGrid = parametersGrid)}) #time 178.74s
median(xgb.lm$resample$ROC) #0.7544
mean(xgb.lm$resample$ROC) #0.74123
sd(xgb.lm$resample$ROC) #0.116

#lm-pls-xgb
set.seed(2020)
system.time({
xgb.lm.pls <- train(overall_ad~., data.lm.pls, method = "xgbTree", metric = "ROC",
                trControl = ctrl_xgb, tuneGrid = parametersGrid)}) #time 127.6s
ggplot(xgb.lm.pls)
median(xgb.lm.pls$resample$ROC) #0.7917
mean(xgb.lm.pls$resample$ROC) #0.7595
sd(xgb.lm.pls$resample$ROC) #0.1185

#lm-pca-xgb
system.time({
```

```r
xgb.lm.pca <- train(overall_ad~., data.lm.pca.l, method = "xgbTree", metric = "ROC",
                  trControl = ctrl_xgb, tuneGrid = parametersGrid)})
ggplot(xgb.lm.pca)
median(xgb.lm.pca$resample$ROC) #0.7454
mean(xgb.lm.pca$resample$ROC) #0.7466
sd(xgb.lm.pca$resample$ROC) #0.1483

#lm-enet-xgb
set.seed(2020)
system.time({
xgb.lm.enet <- train(overall_ad~., data.lm.enet, method = "xgbTree", metric = "ROC",
                  trControl = ctrl_xgb, tuneGrid = parametersGrid)}) #time 46.6s
ggplot(xgb.lm.enet)
median(xgb.lm.enet$resample$ROC) #0.824
mean(xgb.lm.enet$resample$ROC) #0.7918
sd(xgb.lm.enet$resample$ROC) #0.115

#pls-xgb
system.time({
xgb.pls <- train(overall_ad~., data.pls, method = "xgbTree", metric = "ROC",
                  trControl = ctrl_xgb, tuneGrid = parametersGrid)}) #time 68.5s
ggplot(xgb.pls)
median(xgb.pls$resample$ROC) #0.8194
mean(xgb.pls$resample$ROC) #0.8135
sd(xgb.pls$resample$ROC) #0.068

#pca-xgb
system.time({
xgb.pca <- train(overall_ad~., data.full.pca.l, method = "xgbTree", metric = "ROC",
                  trControl = ctrl_xgb, tuneGrid = parametersGrid)}) #time 54.22s
ggplot(xgb.pca)
median(xgb.pca$resample$ROC) #0.7517
mean(xgb.pca$resample$ROC) #0.7344
sd(xgb.pca$resample$ROC) #0.1145

#xgb-full
system.time({
xgb.full <- train(overall_ad ~., data_phen[,c(1, 6:2713)], method = "xgbTree", metric =
                  trControl = ctrl_xgb, tuneGrid = parametersGrid)}) #274.599 seconds
ggplot(xgb.full)
median(xgb.full$resample$ROC) #0.7685
mean(xgb.full$resample$ROC) #0.7477
sd(xgb.full$resample$ROC) #0.1141
```

```r
#ROC and PR curves
evalm(list(xgb.lm.enet, xgb.lm.pls, xgb.lm.pca), gnames=c('Limma-Enet-XGBoost', 'Limma-I
evalm(list(xgb.full, xgb.lm, xgb.pca, xgb.pls),gnames=c('XGBoost', 'Limma-XGBoost', 'PCA

#resamples
resamp.xgb = resamples(list("Enet-XGBoost" = xgb.lm.enet, "PCA-XGBoost" = xgb.pca, "Limn
bwplot(resamp.xgb, metric = "ROC")

#Variable importance
varimp.enet = varImp(enet.lm)$importance %>% filter(Overall != 0)
varimp.enet$metabolite = rownames(varimp.enet)
varimp.enet = as.tibble(varimp.enet)
varimp.enet %>%
    ggplot() + geom_col(aes(x = reorder(metabolite, Overall), y = Overall)) +
  labs(x = "Metabolite", y = "Variable Importance") + coord_flip()
#pls-svc
svc.pls.varimp = caret::varImp(svml.pls, scale = T)
ggplot(svc.pls.varimp, top = 20) + labs(y = "Variable Importance", x = "Metabolite")

#pls-rf
rf.pls.varimp = caret::varImp(rf.pls, scale = T)
ggplot(rf.pls.varimp, top = 20) %>% labs(y = "Variable Importance", x = "Metabolite")

#pls-xgb
xgb.pls.varimp = caret::varImp(xgb.pls, scale = T)
ggplot(xgb.pls.varimp, top = 20) %>% labs(y = "Variable Importance", x = "Metabolite")

#Partial Dependence Plots
partial(xgb.pls, pred.var = c("meta1076"), plot = TRUE, rug = TRUE,
        plot.engine = "ggplot2")
partial(svml.pls, pred.var = c("meta1076"), plot = TRUE, rug = TRUE,
        plot.engine = "ggplot2")

partial(xgb.pls, pred.var = c("meta335"), plot = TRUE, rug = TRUE,
        plot.engine = "ggplot2")
partial(svml.pls, pred.var = c("meta335"), plot = TRUE, rug = TRUE,
        plot.engine = "ggplot2")
```