Detecting Mislabeled Genomics Data

Ali Afzal, Spencer McDonough, Hossain Pazooki

EE 542, Cloud Computing, University of Southern California

Without reliable data, quantitative research is meaningless. In genomics and particularly in cancer research, mislabeled data can be perniciously misleading and lead to irredeemable errors. Given the meteoric rise in the size and complexity of datasets, traditional means of supervision of correct labeling are losing their viability. To combat this common occurrence in data gathering, the precisionFDA portal has initiated a competition to explore the use of various machine learning methods to help identify mislabeled data. We propose a two-step supervised procedure which relies on cloud computing as our method. In both steps, decision tree ensemble methods are used to first filter out the data that are correlated with the metadata and therefore themselves, and then use those data to identify mislabeled samples. In addition, a novel clustering method is proposed to ensure correct filtering. Our filtering method is shown to be useful in improving the performance of the model which predicts the mislabeled dataset. To be able to perform such exhaustive searches through the data, we utilized Spark clusters on AWS, which significantly reduced training time.

Large-Scale Machine Learning, Data Analytics, Filtering, Ensemble Decision Trees, Correlation Clustering

I. Introduction

Since earlier efforts of gene sequencing from Rosalind Franklin days, genomics has played a crucial role in shaping the course of precision medicine. As with the advancement of high-throughput molecular analysis of DNA, RNA sequences and proteins it is now possible to understand the molecular basis of diseases such as cancer, which has led to new discoveries in medicine. This development in generating multidimensional multi-omics data has opened many doors towards research in the field of cancer and the development of initiatives like International Cancer Proteogenomic Consortium. Utilizing this multi-omics dataset by applying complex computational algorithms can lead to appropriate treatment for each individual patient and in finding a better solution for a certain disease type. This can also be utilized to identify relationships between the genome and protein for identification of certain type of diseases and the addition of multidimensional information might increase the chances for localizing the problem. This relationship analysis between genomics data can also help in solving many problems one of which is sample mislabeling.

Sample mislabeling which is unintentional swapping of patient samples or data mislabeling which is swapping of patient omics data is one of the oldest and biggest obstacle in biomedical research. This type of mislabeling can lead misdiagnosis and wrong treatment which might results in severe irreversible and invalid outcomes. This error is most common in large scale multi-omics studies which involve multiple experiments from different location or time periods, which increase the probability of human error. Therefore, the need for software that assists in identifying mislabeled data is increasingly urgent. For this purpose, the Food and Drug Administration has launched a challenge named Multi-omics enabled sample mislabeling correction challenge in collaboration with NCI-CPTAC. [1]

The given data set for the challenge compromises of 160 tumor samples divided equally for test and training. Each sample has a subsequent proteomics (quantification based on protein counting) and RNA-Sequence (quantification based on Fragments Per Kilobase of transcript per Million mapped reads). Ten percent of error has been introduced in the data based on observed patterns, and the mislabeling information is provided. Clinical data containing gender and Microsatellite instability (MSI) status, both binary based, are also provided for the 80 samples. The sum total of features for each of the samples is 21,556, making the dataset unusually large and challenging to work with.

In our approach to solve this problem we faced an issue of processing high dimensional data on a single multicore cpu, so to solve this problem we moved to AWS Spark instance. To make the data useful for prediction we preprocessed it. In our pre-processing phase we lowered the dimension of data by finding the relationship between protein samples and RNA-Sequences using hierarchical clustering. Then data is filtered based on feature importance using random forest and clinical data is used as a target for its prediction. Afterwards, we used modern boosting algorithms to determine which samples are mislabeled.

In the following section we provide background and basis for the procedures we used, followed by results. The appendix includes the source-code for the software we used in addition to the results.

Methods & Procedures

I. *Background*

The presence of mislabeling in the dataset or sample mislabeling can arise due to numerous reasons, generally the presence of mislabeling is common in large-scale multi-omics data. The certainty of error increases with scale, different location of testing facilities and human factor. The presence of mislabeling which can also be referred to as imperfections or noise in the dataset impact the performance of the system in terms of classification accuracy and in building the classifier. Each dataset quality is identified by its attributes and class labels and presence of noise in each factor can lead to erroneous model. So the physical source of noise in datasets is attribute noise and class noise. Attribute noise can occur due to erroneous attribute values, missing values and N/A values. The presence of class noise can be identified by misclassification of attributes or different classes with similarities. This type of noise can be dealt with by eliminating instance which contain class noise. Elimination of class also helps in removing outliers and make classification consistent for the detection of mislabeling samples. [2]

The common factor in identifying mislabeling in dataset is through filtering using classification. Many studies involving mislabeled data has covered the two main methods for mislabeling data. The first one involves the incorporation of adaptability into algorithm so that its enhancement can tolerate noise. The second method involves the filtering of data so that mislabeled data is removed before training the classifier. Our approach on the given multi-omics dataset is close to the second method which shares the common ground with many other researchers.[3] Decision trees, nearest neighbor classifier and support vector machines are widely used algorithms for pattern recognition to form filters. The aforementioned paper used these three algorithms to test the clarity of filters. With the presence of no noise the effect of filtering did not contributed a substantial difference for any of the methods. A baseline accuracy is measured with 0 % noise to be compared. With the inclusion of 20% noise all three methods were able to maintain near baseline accuracy. For noise level above 30% filtering improves accuracy in comparison to single-filter algorithm where 1-NN and decision trees performing better classification. But with the increase in the noise level the ability to maintain accuracy near baseline accuracy decreases. [4]

In paper by Mannes, [5] a new ensemble based detection of mislabeling method is discussed. In this method they are using an ensemble of supervised machine learning models to identify mislabeling. The selected algorithms are Bayesian model, Random Forest, Logistic classifier, a Neural Network and a Support Vector Machine. The identification method for detecting a mislabeling sample is based on the misclassification of a sample from all three algorithms. Misclassification from all the algorithms will tag the sample as suspicious and will be treated as mislabeled. Since the data set that is being used contain mislabeled data, the model is being trained several times using k-fold Cross Validation (k-CV). They were able to reach a precision of 70% on Iris and Touch datasets and recommended that the technique has a high potential for detecting mislabeling on several datasets. On the other hand Hamed and Pang [6] were proposing a kernel based learning algorithm for the detection of mislabeled samples. They are formalizing their method as an optimization problem by distinguishing in comparison to others approaches. They claim to have a higher detection rate and low false detection in comparison to ensemble and nearest neighbor methods.

A comparison between ensemble-based model (majority voting) and filtering is performed in paper by Michael and Rony [7]. Their evaluation is based on 9 learning algorithms used individually and ensembled together on a set of 54 data sets. The filtering method is outperformed by majority voting ensemble filtering as long as noise present in the dataset is not high. Moreover, higher classification is being achieved by using ensemble of learning algorithms in comparison to single learning algorithms for filtering.

As shown by the papers reviewed above, the level of noise that is introduced in our dataset is at a significant level. and will likely obscure the results of any study performed using it. But the dataset differs from others in that the dimensionality of it, as with all genomics datasets, is severely skewed. Almost all method discussed by other authors require the dimensionality to be an order of magnitude lower. Since traditional dimensionality reduction techniques significantly hide information due to naive assumptions about the data, we chose to use the filtering process demonstrated below.

II. *Filtering Process*

As with most machine learning models, it's hard to properly train a model without first cleaning the data beforehand. Mislabeling or corrupt data can lead a ML model to make incorrect predictions. In this section we will explore exactly how we manage to classify and filter our data so we can train our ML model to identify mislabeled samples.

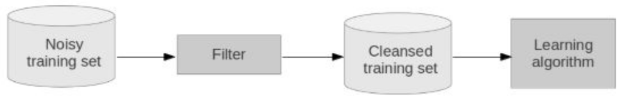


Figure 1. In order to properly train an ML algorithm, the data needs to be cleansed and contextualized.

Typically filtering is done with prior domain knowledge. For example in Kim's [8] work on somatic mutations and their ability to indicate multiple cancer types, the data was cleaned by filtering out specific mutations using prior domain knowledge (whole or partial information of patterns and trends in the dataset). This is useful because it allows the researchers to focus on the unknown variables to find new information using their machine learning model while omitting the variables they know are indicative of other traits.

But what if we don't have prior domain knowledge? We need to be able to sift through our dataset to find the features that will allow us to best predict mislabeling. This is typically where unsupervised learning models come into play. Without any prior information, we could train a neural network, support vector machine, or other models to identify our mislabeled samples. The problem with this, however, is that we do not know how our model is making decisions. These "black box" solutions have the potential to work, but if they don't we do not have the information needed to improve their accuracy. [9] If they do work, then we cannot replicate our results if we want to extrapolate our work to other applications.

Another approach is to create a statistical model using assumptions about the data in order to provide context for our analysis. While this would allow us to completely control the learning process, we would be introducing bias with our assumptions that could influence the model's behavior.

Yet another approach, and the one we have chosen for our research, is to apply random forests to feature classification. As put forward in Carla Brodley's research on identifying mislabeled data on classification, the act of "pre-pruning" decision trees is an effective way of selectively finding the most prominent features that allow us to identify mislabeled data.[10]

This pre-pruning strategy is the process we follow in selecting the best features. Our dataset consists of more than 21000 multi-omics features and 2 metadata fields (gender and MSI, or tumor indicator) across 80 samples. We can train multiple random forests to classify samples based on the data inside to match the target metadata we assign for each forest. In this case, we run two basic forests, one targeted to gender and one targeted to MSI. We then select the most prominent features within the context of gender and msi and use these to train a model to perform the actual sample mislabeling.

In his research on using large ensembles to identify mislabeled data, Mannes Poel uses extensive cross validation to compare the performance of the same model with different hyperparameters.[11] Hyperparameters are what allow a model designer to change attributes of a model such as number of trees in a random forest, depth of each tree, etc. We apply this to our classification model which allows us to reduce the dataset even though it is unusually wide (as are most datasets associated with genomics).

A. Tuning random forests for filtering

As aforementioned, the rationale for choosing decision tree ensembles for our model is primarily the insight gleaned from applying such models. Perhaps neural networks could have been able to slightly outperform such models, but by virtue of their black-box nature, the domain expert that is using this system would be left powerless if the model does not perform on a new dataset.

Random forests, on the other hand, are among the best predictors and they crucially output the feature importances they obtain when learning the model.[12] This allows for further investigation into whether this filtering step was viable, and we suggest the use of correlation clustering to enhance this capability, which we will discuss in the next subsection. The decision as to the use of random forest algorithm over boosting is discussed in the boosting section below.

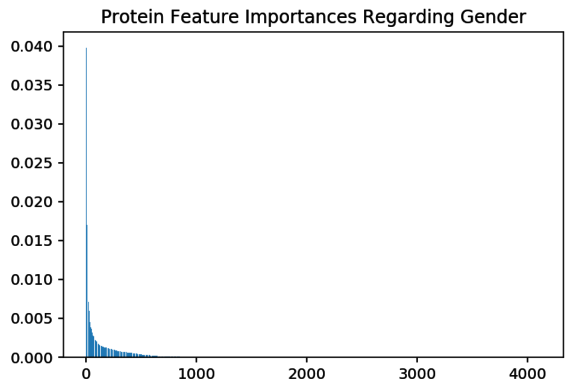


Figure 2. This figure shows how few proteins correspond to gender in the dataset. Each column indicates the importance if each protein (indexed here) with regard to the splits made by the algorithm, the values on the vertical axis sum up this value over the dataset. Due to the extreme long tail of the plot for the entire dataset, the plot is only demosntrated for the 4000 proteomics subsample. The thin line is invisible if all 20,000 features presented.

B. Wide random forests on Spark

The problem with algorithmic tree ensembles, such as random forests has always been the search for the correct hyperparameters. This process requires an exhaustive search for the best combination of the hyperparameters (algorithmic details of model impelementation), and naturally is very computationally extensive. Just as decision trees which are weak classifiers by themselves were augmented to ensemles using computers, the ensembles such as forests and boosting methods are now augmented by the computational capabalitites in the cloud. [13]

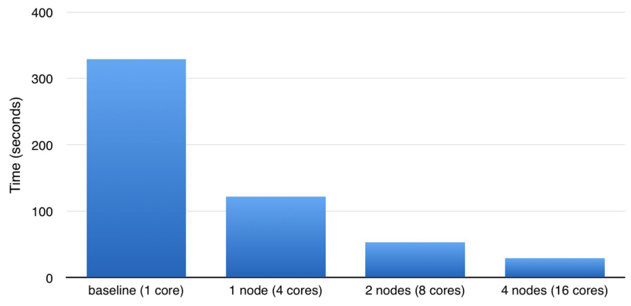


Figure 3. Spark's capability to significanly reduce the time needed for grid search enables many more application of forests in various fields. Here, the nodes and the cores within them are shown to be utilized effectively.

III. *Correlation Clustering*

An alternative approach proposed for filtering metadata is using the unsupervised k-means algorithm first to detect clusters of genes that correspond to given measures in metadata. We found this approach to be problematic given the unusually high dimensions of genomics data. Instead, as a confirmatory step in our filtering process, we use a more recent method, correlation clustering, to confirm the clusters are valid in high dimensional space. This steps ensures the mislabeled samples we aim to identify are easier to detect given the elaborate and methodical filtering steps employed.

In the following parts, we first present the problems of cluster detection in high dimensions, particularly in genomics, and then present our proposed method of dealing with the problem of finding clusters of microarray data that correlate with metadata.

A. *Clustering in genomics*

As was shown in the previous section, current approaches in analyzing DNA microarray heavily emphasize the use of domain knowledge in finding clusters of genes that communicate meaningful information. A study [14] of various methods of prior approaches points to the pitfalls in the use of clustering in analyzing such data. By breaking down the purpose of such studies, the authors address the domain-specific problems that arise in the use of this statistical technique in genomics. The purposes of clustering in genomics often falls under three categories: class comparison, class prediction, and class discovery. Class comparison is the comparison of gene expression in different situations. Class prediction studies are similar to class comparison studies in that the classes are predefined. Class discovery is fundamentally different from class comparison or class prediction in that no classes are predefined.

The authors show that traditional clustering methods, such as k-means, are only suited for class discovery and fail to produce meaningful results for problems of class comparison and class prediction. They attribute this fact to the subjective choice of similarity and distance measures that can used to partition samples into groups. For example, due to the unusin high dimensional cases, standard practices in using clustering methods suggest the use a variety of distance measures in addition to the Euclidean distance measure that is used in two and three dimensional spaces. Some suggest standardizing or adding weights to Euclidean distances, others focus on non-Euclidean measures such as norms or Bray-Curtis to name a few.

Moreover, In genomics we know that only a subset of genes of informative regards any particular characteristic. This domain-specific fact implies that the majority of features and dimensions are just simply noise in most genomics problems. The authors suggest the use of supervised methods that can disregard the unimportant features.

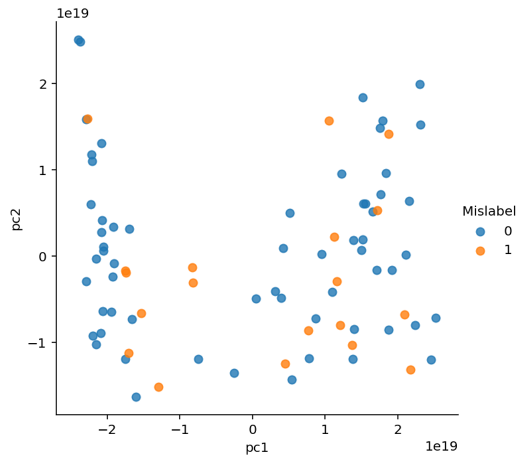


Figure 4. The difficulty of clusteirng is shown here as the two principal components of the filtered data is extracted. As expected, reducing dimensionality is very unlikely to produce useful results.

For the purposes of our problem, our ultimate goal is in fact the discovery of mislabeled samples. But the nature of the problem of mislabeling does not guarantee all the mislabeled samples be contained within distinct classes. Often, the mislabeled samples are not even outliers. Furthermore, DNA microarray by definition include many outliers, since they correspond to estimations of the level of expression of thousands of genes for a sample of cells. For this reason, we chose to break down this problem into two class prediction problems. The first step is filtering for finding the genes that correlate (predict) metadata using large-scale machine learning. And the second is to use machine learning methods to isolate the outliers given the subsample of data chosen. For above reasons, we have used ensembles of decision trees to be able to filter out the irrelevant genes. In the filtering process we leave out the genes that are disregarded by random forest classifier, i.e. the genes that have zero feature importance are filtered out.

To ensure the process of filtering has been indeed successful we use correlation clustering to ensure the genes and proteins identified form meaningful clusters. This is to ascertain the validity of our method, due to the fact that the metadata used for structuring and filtering the data can be mislabeled themselves.

B. Background on Correlation Clustering

The problem of high-dimensionality also arises in natural language processing. In document classification, often the many words contained in a document are irrelevant for determining the meaning and context of blocks of text. A useful method that has recently been proposed and adopted is correlation clustering.[15] Instead of focusing on the samples and measuring the dissimilarity between them to find meaningful groups, an approach that has been proved fruitless, this method proposes to look at feature vectors themselves. Each sample (assuming we have N samples) then becomes a vector with p dimensions (p is the number of features). This flipping of dimensions can be particularly useful in wide datasets where p>>N (where features extremely outnumber number of samples).

Similar to other clustering methods, the similarity measure has to be specified. In this case the options are far fewer, and limited to correlation measures such as linear and rank correlation. After transforming each vector to reflect the correlation measure, the distance between the vectors is measured by the cosine distance between them. These measures are used in turn to construct a dendrogram of hierarchical clusters that show the level of interaction and similarity between features in the dataset. In our case, we chose to use rank correlation because the linear parametric assumptions do not hold in genomics data.

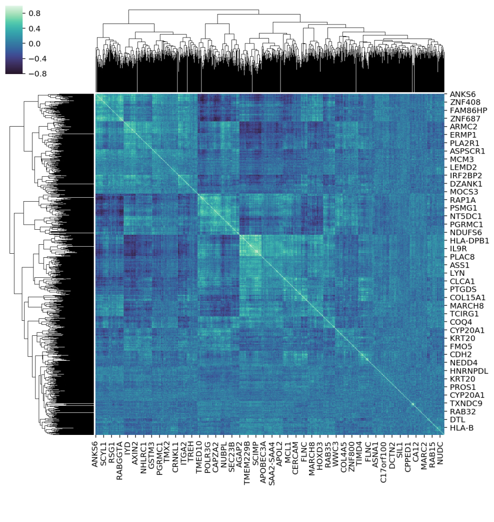


Figure 5. This heatmap shows the result of filtering when the mislabeling noise exists in the filtering process. Although the predictive accuracy is still in the 80% range, the large numbers of genes used and the fact that they are not correlated with oneanother can inform the presence of significant noise in the dataset.

C. Results

The following figures show that the validity of clusters can indeed be confirmed visually. The data is grouped together using hierarchical clustering by using the rank correlation as the dissimilarity measure. The darker colors in the heat-maps correspond to the level of rank correlation between the microarray data, not between the samples. It is evident that dense meaningful clusters of genes can only obtained if the data is not mislabeled. Dense clusters prove to be useful in improving machine learning models that would otherwise be hindered by the high dimensionality of genomics datasets.

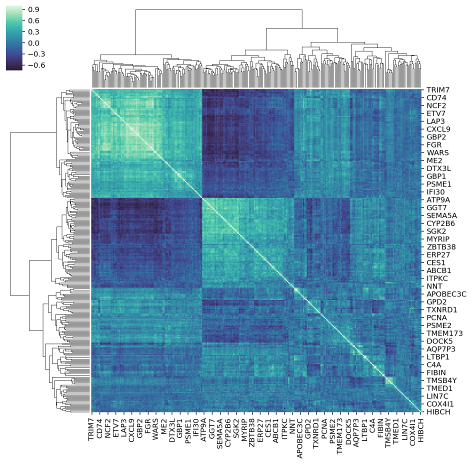


Figure 6. This corrected heatmap shows the correctly filtered outputs. Dark and light features indicate the filtering prcoess has been succesful. Also, less noise has resluted in a more succesful reduction of dimensionality in the dataset, that is reflected in the branches of dendrograms.

IV. Boosting Decision Trees

After proper filtering and preprocessing of the multi-omics data with respect to the gender of the patient and the MSI status of the diagnosed tumor, we now aim to identify the mislabeled samples using the identified genes and proteins that correspond to the metadata. Given the fact that often mislabeled samples are a minority in datasets, the FDA challenge has only set the mislabeling percentage to be 10% of the data in the provided training set, and has indicated which samples are mislabeled. This enables us to take advantage of the mislabeling samples in training our model.

As shown previously, we can make the filtering process more effective by employing a supervised algorithm such as Random Forests. In other words, using the correctly labeled samples for identifying the genes and proteins that correspond to the metadata, we identify the relevant subset of the data that could prove useful in our prediction model. This supervised approach has several advantages: first we do not confuse the exceptions in the data for outliers, second the method will be transferable to other datasets with a methodical approach, and third it is possible to gain insight into the workings of the model by using tree-based algorithms.

Random Forests are ideal for filtering processes of this kind, because they are utilizable out-of-the-box: easily deployed with minimal supervision if sufficient computational power is available for an exhaustive search for the ideal hyperparameters. As we have shown, the advances of cloud computing and distributed implementations of this algorithm have made its use possible for feature selection. But due to the nature of this problem, the final mislabeling detection algorithm has to be able to predict sparse outcomes.

This severe 9-to-1 class imbalance of the correctly-labeled to mislabeled samples renders many machine learning algorithm useless. In our tests, traditional machine learning algorithms were ineffective for one primary reason: efficient optimization is extremely difficult for predicting sparse outcomes. As an example, if correctly-labeled is denoted by 0, and the mislabeled as 1, if the algorithm predicts all 0s the classification accuracy measures (such as the f-1 score, or the random forest out-of-bag measure) will by default be 90% or more. Even if the sparse outcome prediction are weighted using custom scoring metrics, the number of false-positives increased in our experiments as we tried to optimize the algorithms.

Therefore the tools we utilized earlier to use the brute computational force of cloud clusters to obtain relevant information without domain knowledge are not useful in this case. Furthermore, we need an algorithm that has hyperparameters that can be tuned to better predict sparse outcomes. In our research and experiments, modern implementations of Gradient-Boosted trees stood out. Despite their need for extensive hyperparameter tuning and customization, they are ideal for the purposes of the FDA challenge, particularly when the relevant features are selected using a purposeful and elaborate filtering process.

Below we first provide a background on Gradient-Boosted trees, then introduce the library we used and present the results afterwards.

A. Background on gradient-boosted ensembles

Boosting for classification was first introduced in the AdaBoost (Adaptive Boost) algorithm,[16] in which many weak classifiers (in this case a decision tree that predicts marginally better than random) are combined (or boosted) to produce an ensemble classifier with a superior generalized misclassification error rate. The use of Gradient-Boosted algorithms became widespread after its statistical foundations were established.[17] In fact, one of the first successful applications of boosting for predictions was achieved in bioinformatics, in which its superior performance as a predictor was demonstrated compared to other discriminative functions in the classification of tumors using gene expression data.[18]

Contrary to Random Forests, Gradient-Boosting emphasizes additive stagewise learning instead of joint learning the parameters of the model. As with other popular algorithmic methods–neural nets and support vector machines–and traditional statistical models–least squares and maximum likelihood for example– random forest optimizes all of its parameters jointly. This implies that errors in the previous runs are corrected as the learner optimizes its results with respect to the training set, which is the main culprit responsible for overfitting. boosting, on the other hand, the parameters are fit in a stagewise fashion. This slows down the rate at which the model overfits, by not going back to adjust what it had learned in prior runs. In effect, the model is regularized by default. Moreover, boosting learns from the errors of previous trees in the ensemble, albeit it does not correct the loss of them. Whereas Forests are random ensembles of independant (uncorrelated) trees. forest and boosting methods can also be viewed as collection of trees added together with coefficients as in other additive models in statistics. They differ in the fact that Forests by default assign similar weights to all the trees, to reduce bias in the model, but boosting methods can assign different weights to the stumps (one-node trees) if it improves the prediction.

Because of this added level of flexibility, Boosting models outperform forests and many other methods. For example, the majority of Kaggle [19] data science competitions winners are using highly optimized versions of this simple idea. In the next section we present our library of choice for taking advantage of this powerful algorithm.

B. XGBoost library

One the most effective and widely-used machine learning algorithms currently in use is the boosting library as implemented in the XGBoost library, first developed at the University of Washington.[20] XGBoost adds yet another level of regularization to the original gradient-boosters.

XGBoost stands for eXtreme Gradient-Boosting, but as the authors put it, the name was meant to underscore the fact that this boosting method is extremely regularized. The original boosting methods were shown [21] to be analogous to shrinkage methods in regression. The sequence of models produced by a boosted version of linear regression (boosting is performed on the residuals) is similar to the Lasso,[22] which is a regularized model fitting in high-dimensional feature spaces.

XGBoost implements gradient descent for decision tree boosting with an additional custom regularization term in the objective function, for example by adding the L1 norm similar to the lasso or the L2 norm similar to ridge regression. It sequentially adds predictors and corrects previous models but instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction. So in this way, the model is updated using gradient descent and regularized, hence the name.

The primary reason this library has been widely adopted shortly after its release is its efficient implementation. From the start, the boosting algorithm was implemented to be cache-aware and parallelizable. As the authors explain in the main paper,[23] these features were enabled by storing the data into in-memory units, which they refer to as blocks. We discuss this system implementation in the following section.

C. XGBoost distributed system design

Because the main computational bottleneck in tree learning methods is sorting the data, the block-centered architecture proves to be particularly useful in parallelizing this task. In the library, data in each block is first stored in the compressed column format, with each column sorted by the feature value. This formatting is done once before training. Blocks (or column-blocks) can then be distributed among many memory units, be it caches on the same machine or distributed on a cluster.

In finding the best split, traditional boosting algorithms have to scan the entire data to find the best split. This, although accurate, slows down the algorithm since in most cases data is too large to be stored in cache. This so-called greedy approach to split finding is implemented in XGBoost, but the authors of this library also presented an approximate algorithm for split finding that allows the distribution of blocks, and does not require all blocks of memory be accessed at each round of split finding. This makes XGBoost considerably faster than other implementations of gradient-boosting decision tree ensembles.

In addition, the authors underscore the importance of calculating the optimal block size for computation purposes. This cache-aware approach ensures the algorithm optimizes its performance with respect to the context of its implementation. Also, it allows seamless integration with Spark. We show our implementation of this feature in a distributed cluster using the scala language.

D. Tuning XGBoost for predicting sparse targets

Another useful feature in this library is its implementation of customizable hyperparameters. Some are introduced below and the ones we emphasized for the purposes of our model are highlighted.

Two important hyperparameters we took advantage of that were unique to this library was the scaling of positive weights, which allows to add more wight to sparse outputs when splits are being done, and column subsampling, or feature subsampling which is one of the seemingly simple but fruitful steps in preventing overfitting in the model.

Results

As attached in the appendix, our filtering results were outstanding. By taking advantage of exhaustive searches through the hyperparameters, we were able to predict 95% accuracy the proteomics and RNA-sequences that correalte with gender, and with almost perfect accuracy the ones that inform the MSI status of tumors.

Furthermore, by using boosting methods we obtained upwards of 80% accurate predictions of the mislabeled samples, while not increasing the false positive rate of the mislabeled samples.

Conclusion

Overall, the difficulty of outlier detection, and clusteing in higher dimenions is likely to be a significance hindrance to bioinfromatics research. Ideas from other areas of machine learning research are starting to find their way into computational biology. In this paper we showed the efectiveness of two popular methods, forests and boosting, in this context by breaking the problem down into two steps of filtering and prediction. Furthermore, a recent development in natural language processing was shown to be useful in gleaning insight into the workings of genomics dataset.

References

[1] Boja, E, et al. 2018. Right data for right patient: a precisionFDA NCI-CPTAC Multi-omics Mislabeling Challenge. *Nature Medicine*

[2] Zhu, X., & Wu, X. (2004). Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review*, *22*(3), 177–210. https://doi.org/10.1007/s10462-004-0751-8

[3] Guan, D., & Yuan, W. (2013). A Survey of mislabeled training data detection techniques for pattern classification. *IETE Technical Review*, *30*(6), 524–530. https://doi.org/10.4103/0256-4602.125689

[4] Brodley, C., & Friedl, M. (2011). Identifying Mislabeled Training Data, *11*, 131–167. https://doi.org/10.1613/jair.606

[5] Poel, M. (2017). Detecting Mislabeled Data Using Supervised Machine Learning Techniques. In D. D. Schmorrow, & C. M. Fidopiastis (Eds.), Augmented Cognition. Neurocognition and Machine Learning: 11th International Conference, AC 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9-14, 2017, Proceedings, Part I (pp. 571-581). Springer International. DOI: 10.1007/978-3-319-58628-1_43.

[6] Valizadegan, H., Tan, P., & Valizadegan, H. (2007). Kernel Based Detection of Mislabeled Training Examples. Society for Industrial and Applied Mathematics. Proceedings of the SIAM International Conference on Data Mining, 309–309. Retrieved from http://search.proquest.com/docview/1417864940/.

[7] Smith, M., & Martinez, T. (2018). The robustness of majority voting compared to filtering misclassified instances in supervised classification tasks. Artificial

Intelligence Review, 49(1), 105–130. https://doi.org/10.1007/s10462-016-9518-2.

[8] Kim, H. (2018) Pan-cancer analysis of somatic mutations reveals common functional gene clusters shared by multiple cancer types. *Nature Scientific Reports.*

[9] Breiman, L. (2001). Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). Statist. Sci., 16(iss.3), 199–231. https://doi.org/10.1214/ss/1009213726

[10] Brodley, C (1999) Identifying Mislabeled Training Data. *AI Access Foundation and Morgan Kaufmann Publishers.*

[11] Poel, M (2017) Detecting Mislabeled Data Using Supervised Machine Learning Techniques *Springer International Publishing AG*

[12] Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

[13] Bauer, Langit, Luo, Szul and O'Brien, Breaking the "curse of dimensionality" in Genomics using "wide" Random Forests. https://databricks.com/blog/2017/07/26/breaking-the-curse-of-dimensionality-in-genomics-using-wide-random-forests.html

- Figure from Hunetr, Bradley Autoscaling scikit-learn with Spark. https://databricks.com/blog/2016/02/08/auto-scaling-scikit-learn-with-apache-spark.html

[14] Simon et al. Pitfalls in the Use of DNA Microarray Data for Diagnostic and Prognostic Classification, *JNCI: Journal of the National Cancer Institute.*

[15] Bansal, N., Blum, A., & Chawla, S. (2004). Correlation Clustering. Machine Learning, 56(1), 89–113. https://doi.org/10.1023/B:MACH.0000033116.57574.95

[16] Schapire, R., & Freund, Y. (2012). Boosting : foundations and algorithms . Cambridge, MA: MIT Press.

[17] Friedman J, Hastie T, Tibshirani R (2000). "Additive Logistic Regression: A Statistical View of Boosting." Annals of Statistics, 38, 337–374.

[18] Dudoit S, Fridlyand J, Speed T (2002)."Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data." Journal of the American Statistical Association, 97(457), 77–87.

[19] www.kaggle.com/kernels

[20] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining (Vol. 13–17-, pp. 785–794). ACM. https://doi.org/10.1145/2939672.2939785

[21] Hastie, T., Friedman, J., Tibshirani, R., & Springerlink. (2001). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York, NY: Springer New York. https://doi.org/10.1007/978-0-387-21606-5

[22] Ibid 17.

[23] Ibid 20.

Appendix. Software

The notable parts of out source code are presented below. The rest which is widely similar, are documented and attached as jupyter notebook

Scikit-Learn

We used this library extensively particaulrly the classifiers and grid search classifiers. Example of grid search:

```
gender_forest = RandomForestClassifier(n_estimators=500,
                                n_jobs=-1,
                                oob_score=True)
```

Here the classifier is defined and parallelized thoughout the node cores. Out of bag score is requested from the library for comparison purposes

```
sorted(forest.get_params().keys())
```

This method allows us to obtain the changable parameters of the classifier.

```
gender_gs = GridSearchCV(estimator=gender_forest,
            param_grid=[{'min_samples_leaf':[1, 3, 25, 100],
                         'criterion':['gini','entropy'],
                         'max_depth' : [1, 5, 10, 15, 20, 25, 30],
                         'max_features':[None, 'sqrt', 'log2']}],
              scoring='accuracy',
              cv=5,
              n_jobs=-1)
```

Above, defined the grid search variable, and gives it the lists of values requested to be searched. All permuatations of above variables are searched through and the best result with respect to the scoring metric is obtained after five fold cross calidation

```
gender_gs = gender_gs.fit(X_gender_train, y_gender_train)
print(gender_gs.best_score_)
print(gender_gs.best_params_)
```

the best parameters and the best scores are printed after the grid search is performed using the fit methid in sciki-learn library that corresponds to training.

Scipy

The scientific pyton library was used to obtain dendrograms of the obtained features in the filtering process.

```
gender_corr =
np.round(scipy.stats.spearmanr(X_gender_important_train).correlatio
n, 4)

gender_corr_condensed = hc.distance.squareform(1-gender_corr)
```

first the corealtion is obtained by setting the metric to spearmanr which corresponds to rank correaltion

```
z = hc.linkage(gender_corr_condensed, method='average')
```

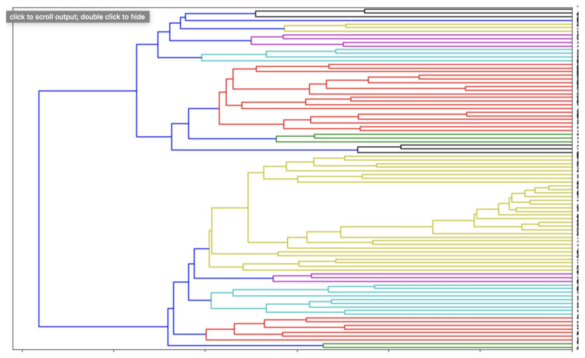
the linkage matrix is constructed using the linkage method in scipy.

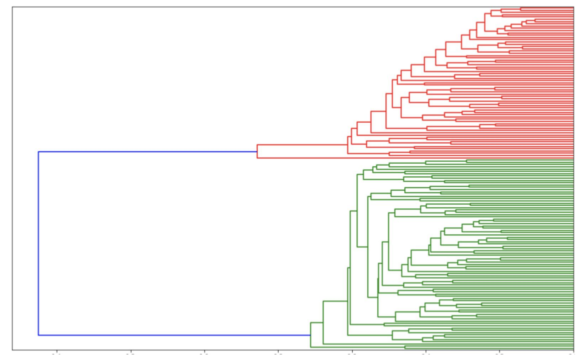
```
gender_dendrogram = hc.dendrogram(z,
labels=X_gender_important_train_df.columns, orientation='left',
leaf_font_size=16)
```

the dendrogram is then created using the linkage matrix defined above.

```
fig = plt.figure(figsize=(16,10))
plt.show()
```

and then the figure is defined and plotted using the matplotlib library. example below. first for gender then msi.





Matplotlib and Seaborn

The two popular plotting libraries were used in conjuction with above to obtain the graphs. Below is an the t-SNE and PCA dimensionality reduction plots obtained with seaborn and sklearn and matplotlib

```
%matplotlib notebook
sns.clustermap(X_train.corr(method='spearman'), center=0,
cmap="mako")
```

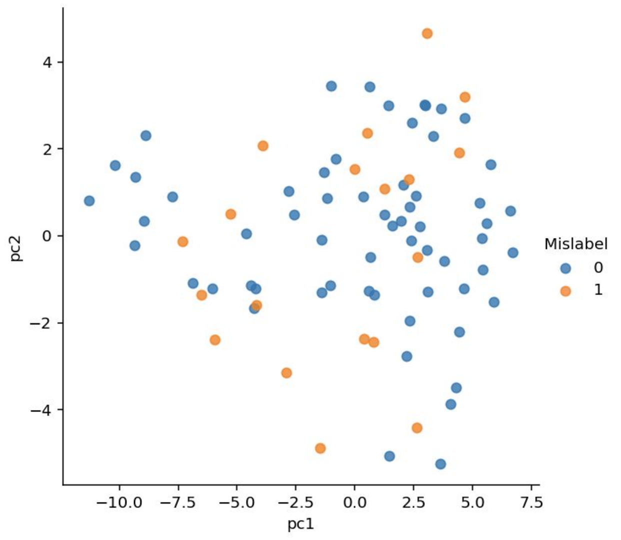
this is the cluster heatmaps presented in the correaltion clusteirng part.

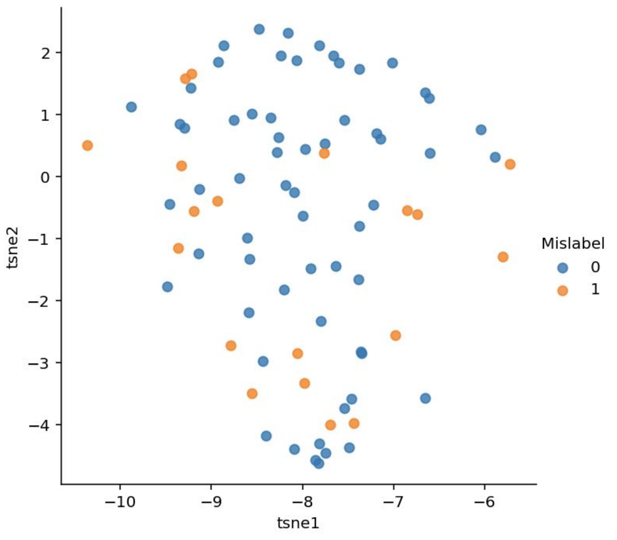
```
kpca = KernelPCA(n_components = 2).fit_transform(X_train)

tsne = TSNE(learning_rate=200, perplexity=40).fit_transform(X_train)

pca_then_tsne = TSNE(learning_rate=200,
perplexity=40).fit_transform(kpca)
```

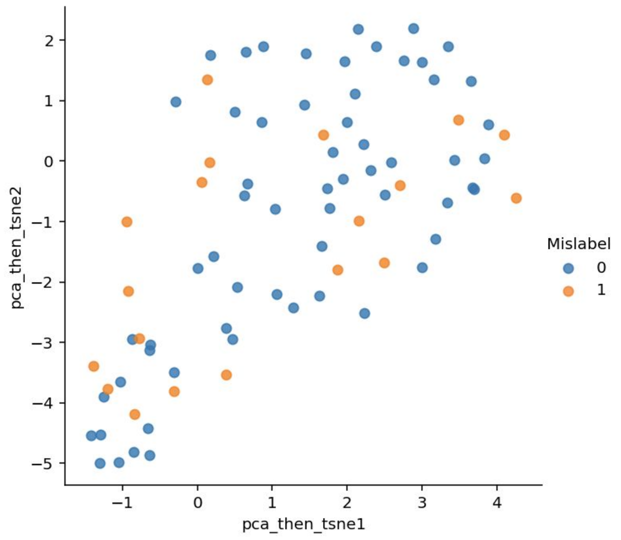
these are the definitons of kernel pca and t-SNE from the sklearn library

```
df_pca = pd.DataFrame(data = kpca , columns = ['pc1', 'pc2'])
df_pca['Mislabel'] = df_train_mislabel['Mislabel']

sns.lmplot(x='pc1', y='pc2', data=df_pca, fit_reg=False, hue='Mislabel',
legend=True)

df_tsne = pd.DataFrame(data = tsne , columns = ['tsne1', 'tsne2'])
df_tsne['Mislabel'] = df_train_mislabel['Mislabel']

sns.lmplot(x='tsne1', y='tsne2', data=df_tsne, fit_reg=False,
hue='Mislabel', legend=True)

df_pca_then_tsne = pd.DataFrame(data = pca_then_tsne , columns =
['pca_then_tsne1', 'pca_then_tsne2'])
df_pca_then_tsne['Mislabel'] = df_train_mislabel['Mislabel']

sns.lmplot(x='pca_then_tsne1', y='pca_then_tsne2',
data=df_pca_then_tsne , fit_reg=False, hue='Mislabel', legend=True)
```

above a pandas dataframe is first defined to contain the principal components (or probabilities in case of t-SNE) with the addition of the milabeling data as the hue needed for seaborn visualization. Seaborn's linear regresison plot (lmplot) is used with the regression fit line (fit_reg) set to false since we want a scatter plot. The plots correspond to each in turn.







XGBoost

As out final predictor, we used this library, the grid search was performe similarly to random forests using the scikit learn wrappers available. Below we show the final prediction part for demostration

```
boost = xgb.XGBClassifier(n_estimators=10000,
                n_job=-1,
                scale_pos_weight=0.9,
                seed=10,
                eval_metric='auc',
                learning_rate=0.3,
                colsample_bytree=0.5,
                max_depth=1,
                subsample=1.0,
               )

boost.fit(X_final_train, y_final_train)
```

the boost classifier is first defined using the sklearn wrapping method. The hyperparameters are as follows:

- n_estimators:
- n_job:
- scale_pos_weight:
- seed:
- eval_metric:
- learning_rate:
- colsample_bytree:
- max_depth:
- subsample:

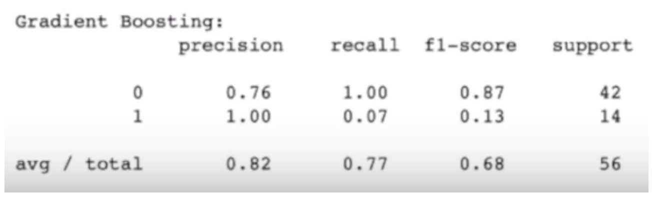
then the booster is trained (fitted) to the final feature matrix with respect to the mislabeling targer vector, the X_train and y_train numpy arrays are obtained using scikit learn's train_test_split method which takes the distribution of classes in to acount when splitting the data

```
X_final_train, X_final_test, y_final_train, y_final_test = \
    train_test_split(X_train.values.astype(int),
                mismatch.values.astype(int),
                test_size=0.7,
                random_state=25)
```

Here the test size is 30% of the data and random_state is used to ensure the data is reproducible.

```
y_final_pred = boost.predict(X_final_test)
print('Gradient Boosting: \n', classification_report(y_true=y_final_test,
y_pred=y_final_pred))
```

then the outcome is printed:



```
Gradient Boosting:
             precision    recall  f1-score   support

          0       0.76      1.00      0.87        42
          1       1.00      0.07      0.13        14

avg / total       0.82      0.77      0.68        56
```

Spark MLLib Code

We used Spark for more efficient hyperparameter search in the filtering process. The code is presented below.

```
val gender = spark.read.
   option("header", "true").
   option("InferSchema", "true").
   csv("s3://fda-proteins/gender.csv")

val msi = spark.read.
   option("header", "true").
   option("InferSchema", "true").
   csv("s3://fda-proteins/msi.csv")

val Array(trainData, testData) = gender.randomSplit(Array(0.8, 0.2))
trainData.cache()
testData.cache()
```

First the data is read in and split into test and train sections. spark.option is used as opposed to sc.option to use the sql features of spark scala implementation.

```
val inputCols = gender_trainData.columns.filter(_ != "gender") // this is
the same for msi

val assembler = new VectorAssembler().
  setInputCols(inputCols).
  setOutputCol("featureVector")

val gender_classifier = new RandomForestClassifier().
  setSeed(Random.nextLong()).
  setLabelCol("gender").
  setFeaturesCol("featureVector").
  setPredictionCol("prediction")

val msi_classifier = new RandomForestClassifier().
  setSeed(Random.nextLong()).
  setLabelCol("msi").
  setFeaturesCol("featureVector").
  setPredictionCol("prediction")

val gender_pipeline = new Pipeline().setStages(Array(assembler,
gender_classifier))
val msi_pipeline = new Pipeline().setStages(Array(assembler,
msi_classifier))
```

Two pipelines are constructed for each of the meatadatam using the vecotirzer and the MLLib classifier

```
val gender_paramGrid = new ParamGridBuilder().
  addGrid(gender_classifier.impurity, Seq("gini", "entropy")).
  addGrid(gender_classifier.maxDepth, Seq(1, 20)).
  addGrid(gender_classifier.maxBins, Seq(40, 300)).
  addGrid(gender_classifier.minInfoGain, Seq(0.0, 0.05)).
  build()

val msi_paramGrid = new ParamGridBuilder().
  addGrid(msi_classifier.impurity, Seq("gini", "entropy")).
  addGrid(msi_classifier.maxDepth, Seq(1, 20)).
  addGrid(msi_classifier.maxBins, Seq(40, 300)).
  addGrid(msi_classifier.minInfoGain, Seq(0.0, 0.05)).
  build()
```

For each of the metdata we constructed a grid search using the paramgrid builder, but the types were incompatible so we had to cast all the columns to double.

```
import org.apache.spark.sql.types.{DoubleType, StringType};

gender_correct.schema.filter(_.dataType ==
StringType).foldLeft(gender_correct) {
   case (acc, col) => acc.withColumn(col.name,
gender_correct(col.name).cast(DoubleType))
}
```

Here, using Scala's idiomatic methods we used foldLeft as opposed to for loop which is less efficient on the spark cluster even though we cached the datasets.

```
val forestModel = bestModel.asInstanceOf[PipelineModel].
  stages.last.asInstanceOf[RandomForestClassificationModel]

forestModel.featureImportances.toArray.zip(inputCols).
  sorted.reverse.foreach(println)
```

Similar to the sklearn method we here print the most important features as obtained by the spark cluster. Output is presented below. The first section of the output is presented below.

```
import org.apache.spark.ml.PipelineModel
import
org.apache.spark.ml.classification.RandomFor
estClassificationModel
forestModel:
org.apache.spark.ml.classification.RandomFor
estClassificationModel =
RandomForestClassificationModel
(uid=rfc_6cb159416dee) with 20 trees
(0.05,RNF20)
(0.05,PSMG1)
(0.05,PRTN3)
(0.05,PADI4)
(0.05,MRPL9)
(0.05,METTL1)
(0.05,MAP2K6)
(0.05,LYPLAL1)
(0.05,IRAK1)
(0.05,IFI35)
(0.05,HYOU1)
(0.05,HSPA2)
(0.05,HMGB2)
(0.05,GGH)
(0.05,DDX50)
(0.05,COPS2)
(0.05,COMMD6)
(0.05,CDC73)
(0.05,C4A)
(0.05,BTK)
(0.0,_c0)
(0.0,ZZEF1)
(0.0,ZYX)
(0.0,ZW10)
(0.0,ZPR1)
(0.0,ZNF706)
(0.0,ZNF638)
(0.0,ZNF326)
(0.0,ZNF280C)
(0.0,ZNF207)
(0.0,ZNF185)
(0.0,ZMYM3)
```

```
(0.0,ZMPSTE24)
(0.0,ZG16)
(0.0,ZFR)
(0.0,ZFPL1)
(0.0,ZCCHC8)
(0.0,ZC3HAV1L)
(0.0,ZC3HAV1)
(0.0,ZC3H4)
(0.0,ZC3H15)
(0.0,ZC3H14)
(0.0,ZC3H13)
(0.0,ZC3H11A)
(0.0,ZBTB7A)
(0.0,ZBED1)
(0.0,ZAK)
(0.0,ZADH2)
(0.0,YWHAZ)
(0.0,YWHAQ)
(0.0,YWHAH)
(0.0,YWHAG)
(0.0,YWHAE)
(0.0,YWHAB)
(0.0,YTHDF3)
(0.0,YTHDF2)
(0.0,YME1L1)
(0.0,YLPM1)
(0.0,YKT6)
(0.0,YIPF6)
(0.0,YIPF5)
(0.0,YIPF4)
(0.0,YBX3)
(0.0,YBX1)
(0.0,YARS2)
(0.0,YARS)
(0.0,XRN2)
(0.0,XRN1)
(0.0,XRCC6)
(0.0,XRCC5)
(0.0,XRCC4)
(0.0,XRCC1)
(0.0,XPOT)
(0.0,XPO7)
(0.0,XPO5)
(0.0,XPO4)
(0.0,XPO1)
(0.0,XPNPEP3)
(0.0,XPNPEP2)
(0.0,XPNPEP1)
(0.0,XPC)
(0.0,XIAP)
(0.0,XAB2)
(0.0,WWC3)
(0.0,WTAP)
(0.0,WRNIP1)
(0.0,WFS1)
(0.0,WDR82)
(0.0,WDR77)
(0.0,WDR75)
(0.0,WDR74)
(0.0,WDR61)
(0.0,WDR5)
(0.0,WDR45)
(0.0,WDR44)
(0.0,WDR43)
(0.0,WDR4
```