Phenotypic Prediction from Transcriptomic Features

Jay Bhatt[1], Faizaan Charania[2] and Jay Torasakar[3]

[1,2,3]Computer Science Department, Stony Brook University

Abstract

Motivation: The output from Salmon, an RNA-sequence mapping and quantification tool, on several datasets is provided. The various samples come from different phenotypes; types of population in this case. The aim of this project is to build a model that takes the Salmon output and predicts the original population/sequencing_center.

Results: Diverse selection and classification models were used on the given equivalence classes. Out of all the models, MultiLayer Perceptron Model was able to generate highest F Score. This model generated an F- Score of 0.89 for classifying into different population and sequencing_center. During the midway report, MultiLayer Perceptron classifier gave a F-Score of 0.91 for classifying into different population

Contact: jbhatt@cs.stonybrook.edu, fcharania@cs.stonybrook.edu, jtorasakar@cs.stonybrook.edu

1 Introduction

The project, provides us with an output from Salmon, an RNA-sequence mapping and quantification tool, on many datasets. The various samples come from different phenotypes. Our main objective here is to generate a model, that takes the Salmon output for a given sample, and predicts the original population/sequencing_center

The baseline model, was set at a score of 67%. The end goal is to achieve the maximum accuracy in classifying the samples using multi-class classification models. A classification technique or a classifier is a systematic approach to building classification models from a given input data set. There are many classification models available, the task is to select the best classification models appropriate for the task. Each technique employs a learning algorithm to identify a model that best fits the relationship between the attribute set and class label for the input data. The model generated by a learning algorithm should fit the input data well and correctly predict the class labels of records.

As a part of this project, different strategies are employed for feature generation, a feature selection algorithm is applied on them and after this preprocessing, models are trained on this data

2 Data: Exploratory Analysis

The data consists of records for 369 accession files that are distributed among seven different population classes and five different sequencing centers. The salmon output for every accession file provides more data about the samples in consideration. Salmon gives us information about all the transcripts that were read for a particular sample. It also gives us information about all the equivalence classes that occurred for every sample and the number of reads that were mapped to every equivalence class.

Every sample has information of approximately 199320 transcripts that map to it. This information by itself provides enough predictive power to classify a given sample with a significantly high f-1 score. Supplementary data about equivalence classes has also been made available. This data provides a more fine grained detail about every sample. There are 1.2 million unique equivalence classes that are observed in the complete dataset. Every sample has reads occurring in approximately 300,000 equivalence classes. The equivalence classes file, provides information about the number of transcripts that make up that equivalence class, the indices of the transcripts in that equivalence class and the number of read mapped to each equivalence class.

Even if only the transcript related features are used to predict the class for a particular sample, the model will have approximately 200,000 features. While data about a high number of transcripts is available, not all of those

transcripts might be relevant in the classification problem. There are some transcripts that might be common across all samples while some that provide significant predictive power. Feature selection techniques have been applied to the dataset to get a subset of richer features that will help the model perform better in classification. The most significant features of the data set will be pivotal in the determining the label of the given sample. Once a good set of features are selected, a model can be fit on those selected features that will give a better score compared a model that would've had to train on all features. Using the complete set of features will decrease the 'signal to noise' ratio for the model and hence help it predict the classes for the samples with a higher accuracy.

The training data set, the accession files are provided along with population and sequencing_center. The population label infers that for a particular accession which population the sample belonged to. Similarly for the sequencing_center.

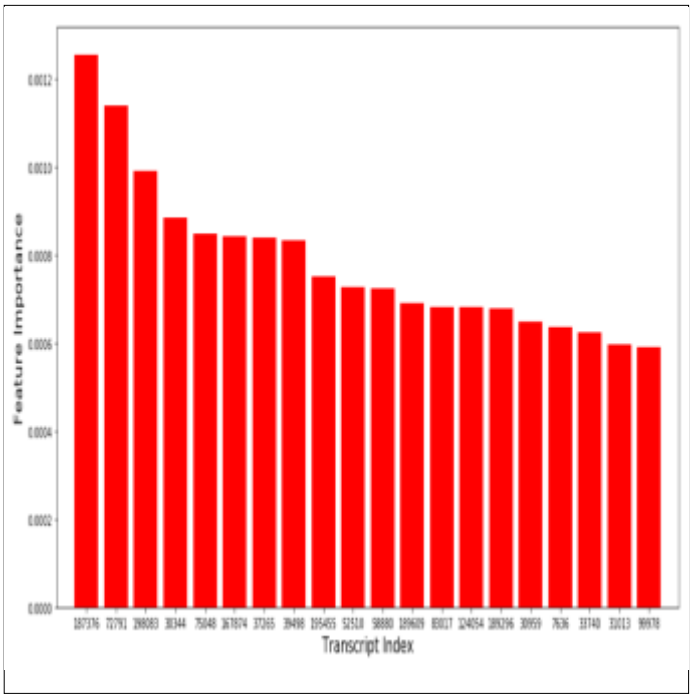

Fig. 1. Feature Reduction

3 Previous Work

3.1 Feature Reduction

In the Midway project, the number of given features were reduced from 200K to 10K using Feature Selection (Extra trees Classifier). Extra Trees Classifier, implements a meta estimator that fits many randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

In the next step, the data is normalized and Z Scores is calculated for each of the feature. Normalization is the process of scaling individual samples to have unit norm.

The next step after Data Preprocessing is the application of Classification Models on the given data. Amongst many classification models, the MultiLayer Perceptron and Random Forest gave the best results amongst all the models. Random Forest gave us a F Score of 0.86 while, MultiLayer Perceptron gave us a F-Score of 0.91. The configuration of the network was:

Input Layer: 10000 nodes
Hidden Layer1: 256 nodes
Hidden Layer2: 128 nodes
Learning rate: 0.005
Tolerance: 0.00001

The F-Score generated for this configuration was 0.91

4 Incorporating Equivalence Class Data

As previously stated, data about equivalence classes provides a more fine-grained information about the samples. There are multiple ways in which these features can be incorporated. The approach followed in this project is to provide the model with as much as relevant information as possible.

Every accession sample has transcripts belonging to approximately 300,000 equivalence classes. When taken a union of all equivalence classes observed, it is found that there are approximately 1.2 million unique equivalence classes observed in the dataset. For every equivalence class that is present for a given accession, we also have the number of reads that were mapped to those equivalence classes. In the previous phase only TPM features were passed to the model, in this phase information from the equivalence classes will also be incorporated in the data.

In this phase, we have attempted to completely transform the feature vector from the TPM dimension to the dimension of equivalence classes. Every feature dimension in this new feature vector will be one equivalence class and the value of the dimension will be the number of reads mapped to that equivalence class for the given accession.

To incorporate the equivalence class data, we make a pass over folders for every accessions and read the equivalence class data. We store this data in the form of a Python dictionary called all_equivalence_classes. The dictionary 'all_equivalence_classes' has accessions as keys and the value is the data about equivalence classes.

We also maintain a master set, which contains data about all the equivalence classes that we have seen so far in the data. It is implemented using a Python set. 'all_equivalence_classes' is a dictionary because for every accession, we have unique information about the presence of equivalence classes and the reads mapped to them.

Every element in the 'all_equivalence_classes' dictionary is another dictionary by itself. This child dictionary consists of equivalence classes as keys and the number of reads mapped to them as values. There are 369 accession folders, each of which consists of numerous equivalence classes. Data about equivalence classes is given in the format
< *'i'* number of transcripts, each of the i transcripts, number of reads mapped to that equivalence class >
To map all the equivalence classes and compare them across accessions, there is a need to uniquely represent every equivalence class. To create a unique id for every equivalence class we concatenate the first two out of three parts of the equivalence data and use the third part as the value. The same unique id is also used as a key in the dictionary that we store under 'all_equivalenc_classes'. While populating data about equivalence classes observed in every accession, we also need to keep updating the master set of all equivalence classes. The master set is implemented using the Set data structure in Python. As the data structure of master set is of the type 'set' it by default only stores unique values of equivalence class IDs. As keep parsing through all the folders, we keep updating the master_set. By the end of the pass, master_set contains unique ids, for all unique equivalence classes.

4.1: Efficiently storing equivalence classes data

As it turns out, storing the keys of equivalence classes as a concatenated string of the number of transcripts in an equivalence class and the IDs of those transcripts was occupying a lot of space in memory. To keep updating the value for all equivalence classes for all accession samples, we need to keep the whole dictionary in memory, which wasn't feasible.

Prof. Rob suggested an approach in which instead of storing the keys as concatenated strings, we store them as int tuples consisting of the number of transcripts in an accession and the IDs of those transcripts. This reduced the memory consumption by a considerable amount. In the next round of improvements, we first concatenate all the data for creating the key in a string and then convert that string to an integer and use that integer as the key for our equivalence classes.

4.1.1 Inspiration from RainbowFish

RainbowFish uses a data structure that minimises redundant representation of similar equivalence classes, by assigning a label to each class and then using that smaller size label to represent the equivalence class. Here the approach was to map the number of reads to its corresponding equivalence class. And to do so we are forming a giant hash table with equivalence classes as key and reads mapped as values. And to reduce the memory consumption, index of an equivalence class in 'master_set' was used to map the large keys to smaller ones.

4.2 Dimensionality Reduction in Equivalence Classes

For the dimensionality reduction of the given feature set, again the Extra Trees Classifier is used, similar to the midway project report. The criteria for choosing the features is that they must have the importance measure > 0.00001. Following that criteria, the number of features were reduced from 1.2 million to about 25000.

5 Multi Task Prediction

5.1 TPM features for Multi Task Prediction

After selecting the top 10000 features, different models were applied to the data.

The following models were taken into consideration:

1. KNN:
    a. KNN algorithm is one of the simplest classification algorithm and it is one of the most used learning algorithms. KNN is a non-parametric, lazy learning algorithm.

    b. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

    c. KNN is a lazy algorithm, meaning either there is no training phase or it is very minimal. So in KNN the training phase is fast. KNN keeps all or most of the training data, which is needed during the testing phase
    d. KNN generated an F Score of 0.66

2. Random Forest Classifier:
   a. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

   b. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

   c. Random Forest is generating an Fscore of 0.81.

3. Extra Tree Classifier:
   a. Extra Tree Classifier implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

   b. Extra Tree Classifier gave a F Score of 0.77

Table 1. Results

Sr No	Model Name	F Score
1	KNN	0.669
2	Random Forest	0.810
3	Extra Tree Classifier	0.77

5.2 Using Equivalence Classes for Multi Task Prediction

After selecting the top 25000 features, MultiLayer Perceptron model was applied

MultiLayer Perceptron:

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function

f: Rm -> Ro

by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features X = $x_1,x_2,....,x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression.

Since Neural Networks are bound to scale well when feature density is high, so MLP was considered for classification.
MLP model generated an F Score of 0.89 which is the best amongst all the models.

Sr No	Target	F Score
1	Multi Task Prediction	0.88257
2	Population	0.80358
3	Sequencing_center	0.93480

6 Implementation details:

Source:
https://github.com/jay-z007/Computational-Biology/

6.1 How to run the Project

A zip of all the pickle files required for running the project is provided with the code.

The compressed file includes:
1.models.pkl:
models.pkl is a dictionary that contains 3 models, one for each, multi-task prediction, population prediction and sequencing center prediction

2. multi_features.pkl,pop_features.pkl,seq_features.pkl:
These files include the top 25000 features to be selected for each model's best performance

3.all_scalers.pkl:
It contains Standard Scaler objects from sklearn that have been fitted on training data

4.binarizer.pkl: Label binarizer, required for multi-task classification part of the project

5.predict,py:
Wrapper class used to test the code on new inputs

6. Command to run:
python predict.py ./models.pkl <address to the test samples root> <address to the test labels file>

Acknowledgements

We would like to thank Prof. Robert Patro for his continuous guidance and suggestions during the implementation of this project. We have not only learnt a lot of new concepts related to computational biology, but have also faced new engineering problems that we can now claim to have experience in overcoming.

References

Almodaresi, F., Pandey, P., & Patro, R. (2017). Rainbowfish: A Succinct Colored de Bruijn Graph Representation. bioRxiv, 138016

http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

http://scikit-learn.org/stable/modules/feature_selection.html

http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html