Classification of regulatory sequences using machine learning techniques

Boris Shilov, Wim Thiels, Lucas Coppens, Zixuan Xie

Introduction

Melanoma is an aggressive cancer with a high level of therapy-resistance due to its high degree of heterogeneity and plasticity. Despite the great efforts put into the field of melanoma treatment, resulting in great advances, many challenges remain. The major challange in fighting melanoma relapse continues to be the tumor heterogeneity, as melanoma comprises a wide variety of phenotypically distinct subpopulations of cancer cells. To be able to adress this issue therapeutically, the underlying mechanisms of heterogeneity need to be characterized. In this paper, a distinction between two major types of melanoma cells is made: invasive and proliferative (Verfaillie et al. 2015; Shannan et al. 2015).

Transcriptional reprogramming of melanoma cells in proliferative state into melanoma cells with invasive characteristics is a critical event at the origin of metastatic spreading of melanoma. Invasive cells have acquired the ability to migrate to other tissues, enter the bloodstream and therefore lie at the basis of the metastatic spreading of cancer in the body. While the transitional mechanisms from proliferative to invasive cancer cell are yet to be characterized more extensively, it is sure that one event lies at the basis of this transition: the transcriptional reprogramming of the cell. Studying the involved genes and regulatory elements using various bioinformatics approaches is therefore a hot topic in the area of melanoma research. Decoding the regulatory landscape could result into the ability to push melanoma cells towards a different cell state, which would be an interesting target from a therapeutic point of view (Verfaillie et al. 2015).

Transcriptomic, open chromatin and histone modification maps of melanoma cultures were constructed, revealing thousands of active cis-regulatory regions, both for proliferative and invasive cells (Verfaillie et al. 2015). It should be possible to discern which cis-regulatory regions are useful for the classification of cell states in melanoma samples if such states are truly distinct in terms of regulatory landscape. The aim of the project was the construction of classifiers predicting whether a regulatory region would be active in proliferative or invasive cell states. Another useful insight that would be gained from these classifiers, is which regulatory regions are the most significant for distinguishing between cell states, thus giving information about the underlying mechanisms and the critical genes and regulatory elements involved in cancer cell state transitions.

Two distinct machine learning techniques were used for the creation of such classifiers, namely the random forests ensemble method and deep learning, making use of convolutional neural networks. Both models were trained on the same training set, which comprises the dataset of active cis-regulatory regions, mentioned hereabove. In this paper, both methods are described and their results are evaluated and compared.

Methods

Random forests

Random forest (RF) is a nonparametric tree-based method that builds an ensemble model from random subsets of features.(Nguyen et al. 2015) RF has shown excellent performance for classification problems, it works well when the number of features is much larger than the number of samples. However, with its typical randomizing mechanism in feature selection, RF models risc having a poor performance when applied to high dimensional data. The main reason for this is that the RF ensemble method uses subset of features randomly sampled from hundreds of features in each iteration. If such a subset is poorly chosen, the choice of nodes for each decision tree is often dominated by uninformative features. This way, trees grown from such a subspace of uninformative features will result in a RF model yielding poor accuracy.(Nguyen et al. 2015) Aiming at improving the performance of the RF model, a feature selction algorithm filtering out the uninformative features before constructing the RF model was proposed. Boruta is an all-relevant feature selection method. It tries to capture all the relevant features that might be relatively important to the outcome variable.(Kursa et al. 2010)

Construction of the feature matrix

As random forests do not have feature learning integrated into the training process. Thus, they require the input data to have labelled features. Before the application of RFs to our data was possible, a way to discover motifs was required, which would then be the features used in training the RF model.

As the raw sequence data was not suitable for training the random forests model, a pipeline to process the sequence data into a usable form, i.e. a feature matrix, was engineered, taking a considerable amount of time. However this investment was justified by the fact that the sequences themselves were not the principal target, but rather whatever recurring motifs would be present.

Motif discovery and scoring

To discover motifs in the input fasta sequences, the motif discovery utility Hypergeometric Optimization of Motif EnRichment (HOMER) was used, providing a Perl module `findMotifs.pl` that allows for the discovery of enriched regulatory motifs in one set of sequences as compared to another set (the background) (Heinz et al., n.d.). As the goal was to compare and contrast the invasive and proliferative cell states, these two sets of sequences were ran as backgrounds against each other, identifying the most differential motifs. The results of these two runs can be found in `HomerOutput`. HOMER analysis served two purposes: it found known motifs and discovered some *de novo* motifs. The motifs returned by HOMER were formatted as position weight matrices (PWM), representing the relative frequencies of each nucleic acid on each position in the motif.

Having identified the ensemble of differentially enriched motifs in these two cell states compared to each other, the Cluster-Buster software was used in order to score the invasive and proliferative sequence sets for each motif, independently (Frith, Li, and Weng, n.d.). This resulted in a log-likelihood score for every motif against every sequence in the sequence sets. The functions for this step of the analysis can be found in `motif_processing_main.sh`. The set of sequences scored for every motif was then assembled into a feature matrix. In such a feature matrix, every row represents a genomic region and every column represents a particular motif, hereafter referred to as features, with the log-likelihood scores as values in this matrix. Furtherly, a `_label` column representing the binary outcome variable was inserted, with value 0 representing the invasive cell state and value 1 representing the proliferative cell state. This step of the analysis can be found in `feature_matrices.py`, utilising the `feature_matrix_special` method of the `cbust_result` class, created in order to handle the output of Cluster-Buster. Thus, a complete data matrix was produced, formatted correctly to be used as training data for a random forests classifier.

Feature Selection

The resulting feature matrix contained 333 features. As mentioned above, RF classifiers tend to perform poorly when trained on datasets containing too many features. Thus, methods to reduce the amount of features in the training data for our classifier were required. Two software packages were used in order to obtain a reduction of the amount of features, called MAST and Boruta.

MAST

The model-based Analysis of Single Cell Transcriptomics (MAST) R package provides methods and models for handling zero-inflated single cell assay data (McDavid et al. 2018). This packages wasn't originally created for the feature selection of feature matrices for learning purposes. However, it provides a method `LRT` that performs a likelyhood ratio test for single cell assay objects. When reformatting the feature matrix to a single cell assay object, this likelyhood ratio test can be used to detect the most significant features differentiating between invasive and proliferative cell states. The R script used for this reformatting and running the likelyhood ratio test can be found in `Mast_reduce.R`. After running the likelyhood ratio test on the feature matrix for every motif, the most significantly differentiating features (P-value lower than 0.05) were selected. The resulting feature matrix contained 164 feature columns.

Boruta

Boruta is an all-relevant feature selection method, aiming at finding relevant features that are suitable for the prediction of the outcome. (Daniel Homola 2017) It seems to be similar to MAST, however, it has a different test mechanism to discover relevant features. First it duplicates the dataset and makes the values shuffling in each column which after that the duplicated dataset is called shadow features. Then training a RF classifier on the dataset, from which the importances for each of the features are generated. Boruta algorithm involves several iterations (default = 100). It trains the original dataset along with the shadow features dataset and checks for each of the real features if they have higher importance than the best of the shadow features in each of its iteration. If they do, boruta records this in a vector called a hits and continue with another iteration. Finally after the predefined set of iterations it ends up with a table of these selected features. The python implementation in `boruta_reduction.py`

Construction of the model

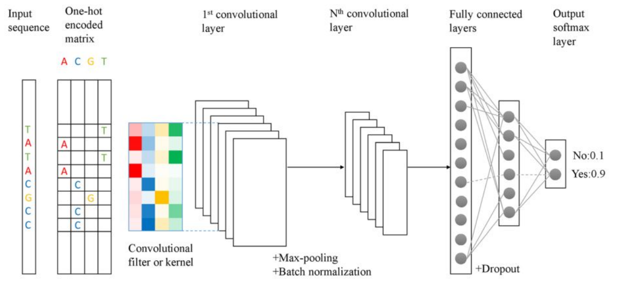
For the construction of the random forests ensemble model, the python package `scikit-learn` was used, more specifically, its class `sklearn.ensemble.RandomForestClassifier`, which offers a standard RF classifier. The classifier was constructed using following parameters: `n_jobs=-1`, which means using all processors to work in parallel for both fit and predict; `n_estimators=1000`, This is the number of trees in this forest, higher number of trees gives better performance but makes fitting slower; `max_depth=None`, the maximum depth of the tree, None means nodes are expanded until all leaves are pure or until all leaves contain less than 'min_samples_split samples'; `min_samples_split=2`, minimum number of samples required to split an internal node (default=2); `class_weight='balanced'`, weights associated with classes, "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).(Srivastava et al. 2015)

BORIS MAYBE HERE YOU COULD TAKE A LOOK AND CHECK ANYTHING ELSE SHOULD BE ADDED?

Deep learning

Two architectures

A schematic outline of how a neural network can be used in motif detection is given in figure ??. By converting the sequence into a 1-hot vector, it becomes an image-like input which can be scanned with a number of kernels from the first convolutional layer. This information is then passed on to the rest of the network. During training, the network will adjust its weights in order to optimize the classification task. In the process, the kernel weights of the first convolutional layer will start to resemble the various DNA motifs which are then extracted and compared to a motif database (JASPAR). (Min et al. 2017)



In our case, 2 model architectures were compared (see fig??). Model 1 uses 3 CNN's, Model 2 uses only 1

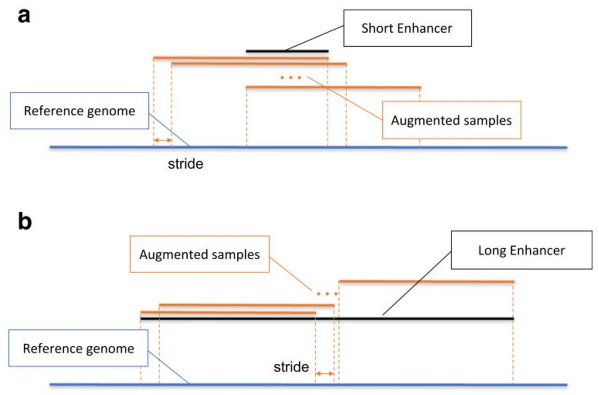
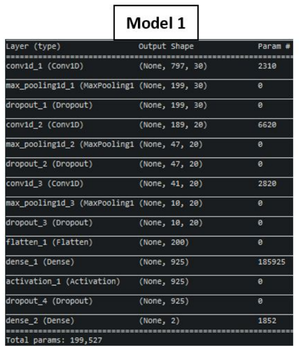
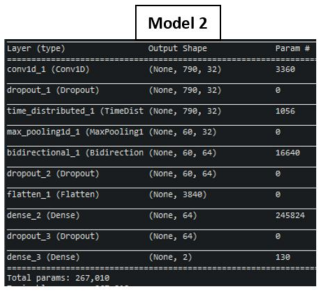


Figure 1: Diagram of data augmentation. By using a sliding window approach, each enhancer sequence produces multiple sequences of equal length that can be used as input for the first convolutional layer. Taken from Min, Xu et al (2017)

convolutional layer, and incorporates a recurrent layer (LSTM's). A recurrent neural layer can pick up on sequential information and are therefore better suited to capture the context of the motifs (motif syntax).


Model 1

```
Layer (type)                 Output Shape              Param #
==============================================================
conv1d_1 (Conv1D)            (None, 797, 30)           2310
max_pooling1d_1 (MaxPooling1 (None, 199, 30)           0
dropout_1 (Dropout)          (None, 199, 30)           0
conv1d_2 (Conv1D)            (None, 189, 20)           6620
max_pooling1d_2 (MaxPooling1 (None, 47, 20)            0
dropout_2 (Dropout)          (None, 47, 20)            0
conv1d_3 (Conv1D)            (None, 41, 20)            2820
max_pooling1d_3 (MaxPooling1 (None, 10, 20)            0
dropout_3 (Dropout)          (None, 10, 20)            0
flatten_1 (Flatten)          (None, 200)               0
dense_1 (Dense)              (None, 925)               185925
activation_1 (Activation)    (None, 925)               0
dropout_4 (Dropout)          (None, 925)               0
dense_2 (Dense)              (None, 2)                 1852
==============================================================
Total params: 199,527
```


Model 2

```
Layer (type)                 Output Shape              Param #
==============================================================
conv1d_1 (Conv1D)            (None, 790, 32)           3360
dropout_1 (Dropout)          (None, 790, 32)           0
time_distributed_1 (TimeDist (None, 790, 32)           1056
max_pooling1d_1 (MaxPooling1 (None, 60, 32)            0
bidirectional_1 (Bidirection (None, 60, 64)            16640
dropout_2 (Dropout)          (None, 60, 64)            0
flatten_1 (Flatten)          (None, 3840)              0
dense_2 (Dense)              (None, 64)                245824
dropout_3 (Dropout)          (None, 64)                0
dense_3 (Dense)              (None, 2)                 130
==============================================================
Total params: 267,010
```

Both models were built using the python Keras package on a Tensorflow backend.

Data augmentation

Even though a neural network can work with raw DNA sequences as input, there are still two issues that need to be overcome. First, the input to the first convolutional layer needs to be of fixed length, and second, a deep neural network needs a vast amount of training data. To overcome these 2 issues, a moving window approach as in (Min et al. 2017) (with stride = 20) is used (fig ??).

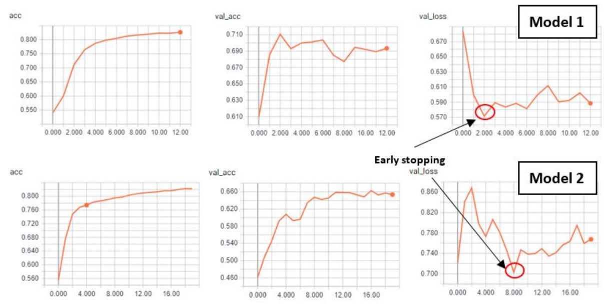


Figure 2: Training metrics for both models

Training and validation

Input matrix creation

Starting from 20122 enhancer sequences, 228051 sequences are generated using the data augmentation approach. The sequences are randomly split in a 70/20/10 fashion (train/test/val). The loss function was class weighted to counter the fact that the I-label is overrepresented in the input data. Both the validation and test data were also balanced.

Feature filtering

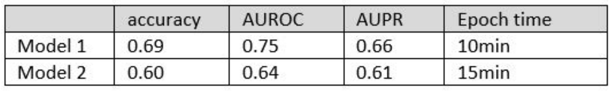
Motif extraction

After training, the kernels of the first convolutional layer, can be screened for motifs. To go from the kernel weights to a proper PSW matrix for a motif, each of the kernels are scored against each of the original DNA sequences. This score represents the best match of a motif when slided across a particular sequence. For each motif, the top 100 scoring sequences are stored, and these 100 sequences are then summarized into 1 PSWM. These PSWM's are then queried against the JASPAR motif database using TOMTOM [2]. Only motif matches with a threshold of E < 0.01 are considered significant.

Results and discussion

Random Forests

The trained classifier was tested on a balanced test set of 3 994 sequences (20% of the total). The performance was evaluated using two different measures: the AUC of the resulting ROC curve and its accuracy, calculated as the total amount of correctly predicted positives and negatives divided by the total amount of predicted sequences. The resulting AUC was about 0.67, as can be seen in Figure??.



	accuracy	AUROC	AUPR	Epoch time
Model 1	0.69	0.75	0.66	10min
Model 2	0.60	0.64	0.61	15min

Figure 3: Comparing classification performance of both models

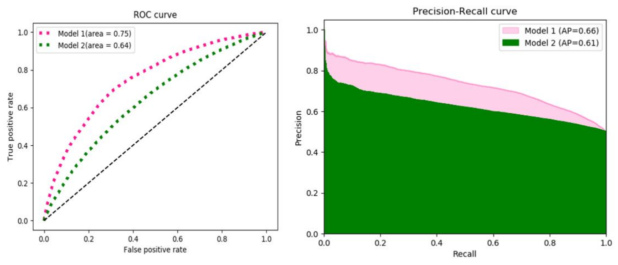


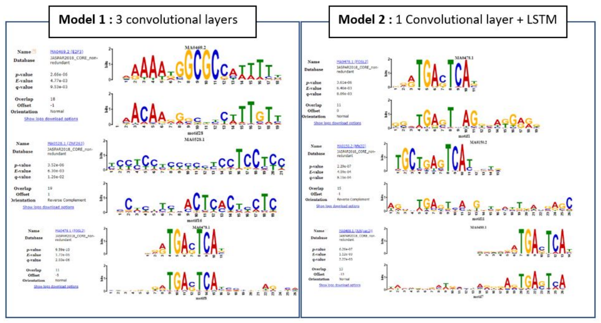
Figure 4: ROC and Precision Recall curves comparing both models

Deep learning

###Comparing classification performance

Both models are tested on a balanced test set of 45333 sequences (20% of the total). From Table1 and Figure?? it is clear that model 1 outperforms model 2. The model using multiple convolutional layers achieves an accuracy that is 9% higher then the model using the recurrent layer.

###motifs detected The motifs detected by both models are shown in fig ??


Model 1 : 3 convolutional layers
Model 2 : 1 Convolutional layer + LSTM

References

Daniel Homola. 2017. "Boruta 0.1.5." 2017. https://pypi.org/project/Boruta/.

Frith, Martin C, Michael C Li, and Zhiping Weng. n.d. "Cluster-Buster: Finding Dense Clusters of Motifs in Dna Sequences." *Nucleic Acids Res* 31 (13): 3666–8.

Heinz, Sven, Christopher Benner, Nathanael Spann, Eric Bertolino, Yin C Lin, Peter Laslo, Jason X Cheng, Cornelis Murre, Harinder Singh, and Christopher K Glass. n.d. “Simple Combinations of Lineage-Determining Transcription Factors Prime Cis-Regulatory Elements Required for Macrophage and B Cell Identities.” *Mol Cell* 38 (4): 576–89. https://doi.org/10.1016/j.molcel.2010.05.004.

Kursa et al. 2010. “Feature Selection with the Boruta Package.” *Journal of Statistical Software*.

McDavid et al. 2018. “MAST: Model-Based Analysis of Single Cell Transcriptomics.” 2018. https://bioconductor.org/packages/release/bioc/html/MAST.html.

Min et al. 2017. “Predicting Enhancers with Deep Convolutional Neural Networks.” *BMC Bioinformatics*.

Nguyen et al. 2015. “Unbiased Feature Selection in Learning Random Forests for High-Dimensional Data.” *The Scientific World Journal*.

Shannan et al. 2015. “Heterogeneity in Melanoma.” *Cancer Treatment and Research*.

Srivastava et al. 2015. “Tuning the Parameters of Your Random Forest Model.” *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/.

Verfaillie et al. 2015. “Decoding the Regulatory Landscape of Melanoma Reveals Teads as Regulators of the Invasive Cell State.” *Nature Communications*.