Predict disease classes using genetic microarray data

Tianye Zhao

August 14, 2020

1 Steps

Here we use *python* with *pandas*, *numpy*, *scipy*, and *sklearn* libraries.

1.1 Data cleaning

Use *pandas* library to read data and use *numpy* library to remove any genes with value ≥ 16000 and ≤ 20 in both train and test data. Through value counts of classes, we know the train data is imbalanced.

MED	39
EPD	10
RHB	7
MGL	7
JPA	6

1.2 Top genes

Remove genes in train data with fold difference ≤ 2. Use *scipy* library to run t-test for each class to select top genes.

1.3 Find best

Use *sklearn* library label encoder to transform classes for further easy manipulation, cross validate the training sets, and grid search the optimal hyper parameter after found best gene set.

- Naive Bayes: we compared Gaussian, Multinomial, Complement, and Bernoulli to find the best distribution for train data.
- Decision Tree:we change max depth and pass class weight to improve the result as train data is imbalanced.
- K-NN: test on $K = 2, 3, 4$, and further in different weight function.
- Neural Network: change hidden layer size and activation function to see any improvement.
- Random Forest: change max depth and number of trees in forest. Moreover, as we had seen the improvement of class weight in Decision Tree classifier, here we directly pass it to improve the accuracy.

1.4 Prediction

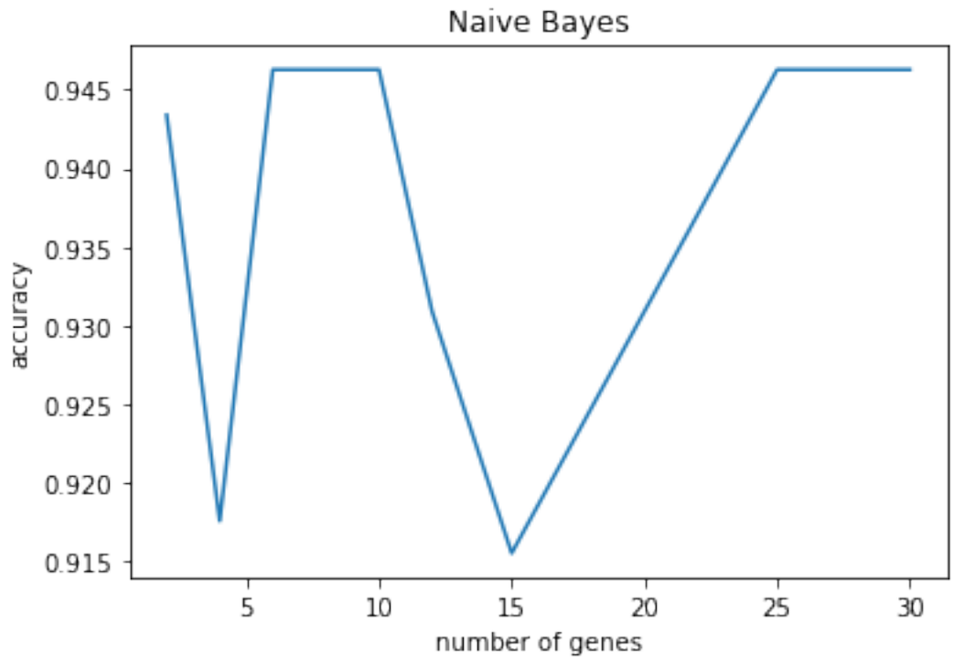
Use random forest and top 6 gene set to make prediction and generate classification report.

2 Classifiers

Here we use classifiers in sklearn library.

2.1 Naive Bayes

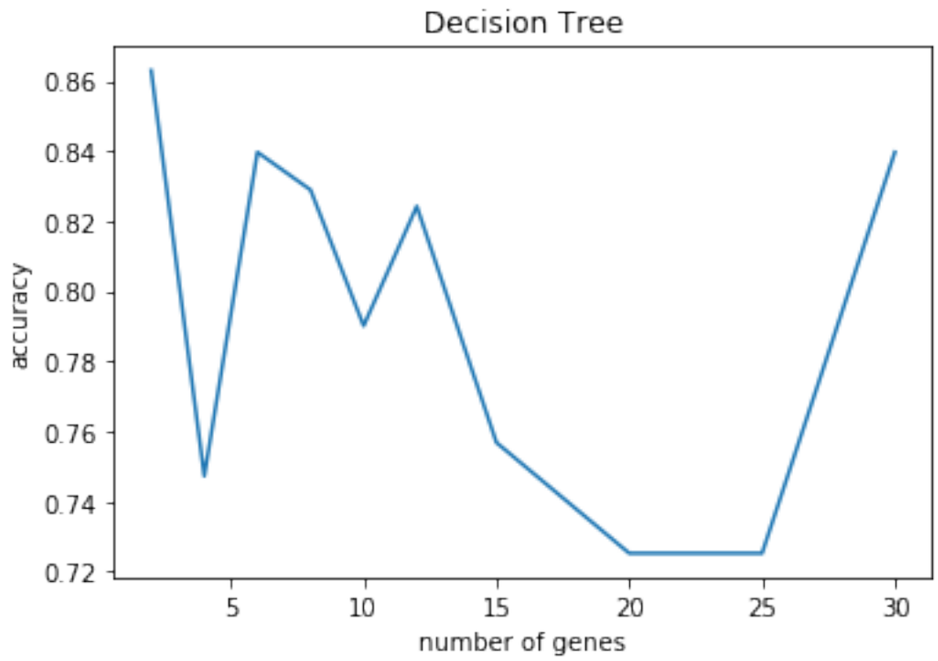
Naive Bayes is to allow all attributes to make equally important and independent contributions towards the decision. It uses Bayes' theorem. Missing values can just be omitted from the calculation. It might use Gaussian distribution to estimate probabilities for numerical attributes. It rivals and sometimes outperforms more sophisticated classifiers on many datasets. It is effective in classification problems because it is sufficient for the correct class to receive the greatest probability. However, it performs poorly when redundant attributes are present.




```
from sklearn.naive_bayes import GaussianNB
```

2.2 Decision tree

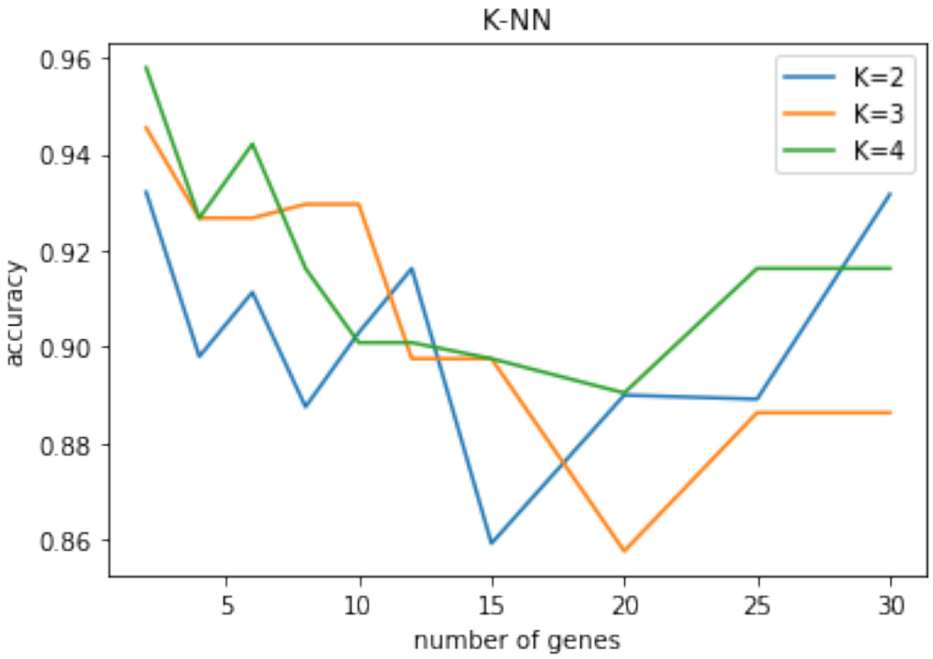
It uses divide and conquer approach. Each node tests a given attribute, comparing it to a constant or nominal attribute. Leaf nodes give a classification that applies to all instances that reach the leaf. Unknown instances are routed down the tree according to value of attributes until a leaf is reaches. For numerical attributes, it can use two-way or three-way split. It might use imformation gain to measure purity of split. The goal is to split in a way to reduce the uncertainty as much as possible. Ideally process will terminate when all leaf nodes are pure.




```
from sklearn.tree import DecisionTreeClassifier
```

2.3 K-NN

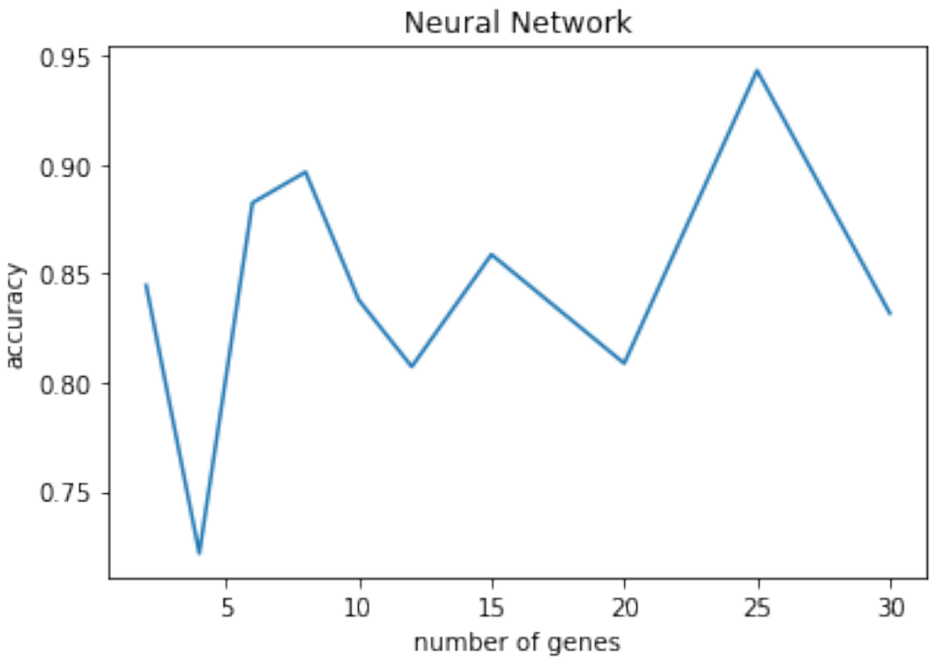
Instance based learning works by storing training examples, and classifying new instances based on distance from training examples. It is lazy learner as it waits for new instances and performs computation at classification time. It is a non-parametric method, it works well even when decision boudary is irregular. One of the advantages of instance-based learning over other machine learning methods is that new examples can be added to training set at any time.




```
from sklearn.neighbors import KNeighborsClassifier
```

2.4 Neural network

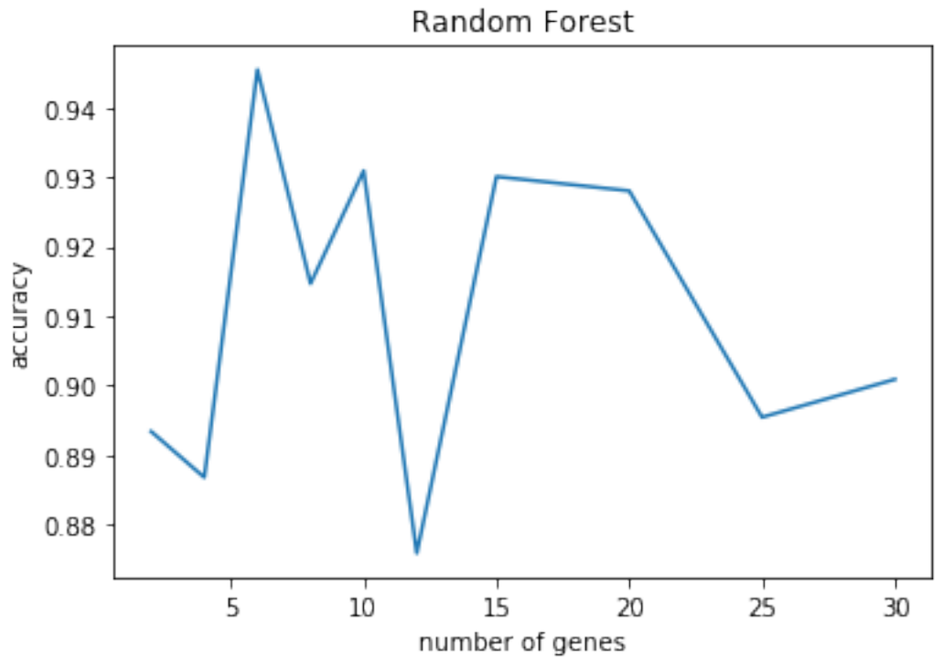
It connects many simple perceptron-like models in a hierarchical structure to perform nonlinear classification. Any expression from propositional logic can be converted into a multilayer perceptron, because AND, OR, and NOT can be represented by single perceptron. Individual units can be combined together to form arbitrarily complex expression. Backpropagation is a simple algorithm for determining the values of the weights. Learning the structure of network is complicated.




```
from sklearn.neural_network import MLPClassifier
```

2.5 Random forest

Random forest is an ensemble learning method for classification and regression. It constructs a multitude of decision trees using bagging and is created by combining bagging and randomization. It builds a randomized decision tree in each iteration of bagging algorithm. It selects random subset of features to split on at each node and combines predictions using mode to generate classification.




```
from sklearn.ensemble import RandomForestClassifier
```

3 Conclusion

Finally, we use Random Forest classifier and choose top 6 genes set. Because it gives us the highest accuracy – about 0.97 mean-test-score, and we show it correctly predicts all the instances in train file in classification report.

	precision	recall	f1−score	support
EPD	1.00	1.00	1.00	10
JPA	1.00	1.00	1.00	6
MED	1.00	1.00	1.00	39
MGL	1.00	1.00	1.00	7
RHB	1.00	1.00	1.00	7
micro avg	1.00	1.00	1.00	69
macro avg	1.00	1.00	1.00	69
weighted avg	1.00	1.00	1.00	69

4 Comment

4.1 Strengths

Decision tree induction is an unstable process, slight changes in training data can easily result in different attribute chosen at some node. Random forest use bagging to combine the trees by having them each vote on a given test instance's class, so more trees lead better prediction. Moreover, different weight initialization may yield different end results. To make classification outcome more stable, random forest runs leaner several different times with different random number seeds, combine results with voting. It randomly picks from the top N options to split in decision tree.

4.2 Weaknesses

While in random forest, each model is built separately because of bagging, each model is not influenced by the performance of previously built models. It gives equal weight to all models, doesn't take previously incorrectly handled instances into consideration.