COSC 3337 Assignment 2:
Group 5 Report

By Colby Kuhnel, Niels Moeller, Gal Egozi, and Raymond Sutrisno

I. Introduction

The purpose of this assignment was to investigate the performance of a selection of commonly used machine learning algorithms and apply them to one of a selection of standard data sets to compare their performance. The datasets available to use were:

- Image Segmentation
- Molecular Biology
- Activity Recognition

Of these we chose dataset 2 which the molecular biology dataset. The machine learning algorithms we used were Neural Networks (done by Gal Egozi), Support Vector Machines (done by Colby Kuhnel), Decision Trees (done by Niels Moeller), and Random Forest (done by Raymond Sutrisno).

II. Algorithms

This assignment required 4 machine algorithms to be used. Of the four, two were required: Support Vector Machines and Neural Networks. The remaining two were free for us to choose. For that we chose Random Forest and Decision Trees.

Neural Network is a method for machine learning that has multiple nodes and layers of nodes to achieve high accuracies in regression and classification tasks. Each node is a combination of a linear function whose output is passed to a non-linear activation function.

Support Vector Machines are discriminative classifiers defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm produces an ideal hyperplane which categorizes new features.

Decision Trees are a non-parametric supervised learning algorithm used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Random Forest is an ensemble learning algorithm that uses a forest of trees as the ensemble. The forest is diversified so to speak by different distribution methods of training examples to them such as bagging or boosting. It is found to be better in general to have populous forests of saplings than few numbers of tall trees.

III. Dataset Preprocessing and General Training Procedure

The molecular biology dataset is a supervised dataset which maps a sequence of 60 DNA base pairs to classes relating to the RNA splicing. These classes are ‘EI’ for Exon/Intron boundaries, ‘IE’ for intron exon boundaries and ‘N’ for neither. The machine learning problem the dataset suits the most is a typical pattern recognition problem.

The sequences are given as 60 character long strings. Additionally there is a third piece of information known as the source which is information about where this sample was extracted from. We chose to ignore this as it we were more interested in this data to benchmark machine

learning algorithms and not for bioinformatics purposes. The class distribution of the dataset is given in the table below:

# of EI class samples	767
# of IE class samples	768
# of N class samples	1655
Total # of samples	3190

The data is categorical and not of real numbers so no special normalization is needed. Additionally since the class distribution is not highly imbalanced, no special class imbalance algorithms such as over or under sampling are needed.

Class Types	EI, IE, N
Feature Types	A, T, C, G, D, R, S, T

Since the features were given as one block, some preprocessing is needed in order to use this dataset. The first step was to turn the features from a single string 60 characters long to an array of 60 characters. Then for implementation reasons and because of the categorical nature of the features and the “one vs. all” learning algorithms used, each feature in this array of 60 characters must be one-hot encoded. After doing a range check of all possible values each base pair can be we found that 8 possible base pairs. After one-hot encoding each training example in total contains 480 elements usable by the algorithm.

In the end our data set consisted of 3190 rows, 480 feature columns and 1 class label column. For Neural networks the feature column had to be one hot encoded further which created 3 columns for class. The reason why classes did not have to one-hot encoded for SVM, Decision Trees, and Random Forest is implementation specific to the library we used for them which is discussed in the next section.

For general training, tuning, and testing we opted to perform three different runs of each algorithm. These runs differed by the ratio of examples reserved for training/tuning and testing. The ratios were, given as training to testing, 80:20, 50:50, and 30:70. These ratios were chosen arbitrarily. Of the examples set aside for training, 10 fold cross validation was used to select for the best model out of a small configuration space of model parameters. The specifics for each model are given in *Section IV Algorithm Specific Experimental Procedure and Implementation Details*

IV. Algorithm Specific Experimental Procedure and Implementation Details

To implement this experiment we chose to use Python as our language platform of choice as it is a lingua franca among all of us and more familiar to us than R. The full list of package dependencies is given as a requirements.txt file in the source code.

For general purpose data preprocessing we used popular libraries such as numpy for matrix manipulation and pandas for dataframe manipulation and organization.

For the machine learning models, metrics, and other utilities for preparing training data we used scikit learn and keras. The algorithms implemented by scikit learn were SVM, Decision Trees, and Random Forest. Additionally for these scikit-learn implemented algorithms, gridsearch was used for model parameter tuning. The cross validation used by grid search was not stratified but instead simple random sample. Of important note to mention is that due to scikit learn's implementation of these three algorithms, they did not require the labels to be one-hot encoded but only the features.

Keras was used for the Neural Network implementation because it was a reasonably good library. Unlike Tensorflow, it is known for its ease of use and has an option for a backend that supports AMD GPUs. Also, scikit learn does not support neural networks.

A. Neural Networks

A neural network was created using the sequential object from keras. Then each layer is added one-by-one using the add function, with a dense function specifying the type of layer. The dense function has three parameters, one for the number of nodes, one for the activation function, and one for the input dimension shape. Next, the model is compiled and fitted. Finally, the model is evaluated. The code looks something like this:

```
model = Sequential()
model.add(Dense(50, activation= 'relu', input_dim=480))
model.add(Dense(3, activation= 'sigmoid'))
model.compile(loss='categorical_crossentropy', optimizer= 'adam', metrics=['accuracy'])
```

B. SVM

We can create a SVM model using the SVC function and the parameters it has for tuning. Here is an overview of the parameters and their defaults:

```
class sklearn.svm. SVC (C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape='ovr', random_state=None)
```

Additionally along with the estimator, we can pass a parameter set to the GridSearchCV and cross validate 10 folds with any combination of the parameter set to find the best fit. This is particularly ideal not only because we can find the best fit estimator, but also because this dataset is difficult to visualize on a graph and recognize the boundaries between points of data. Here is an example of a parameter set for SVC:

```
[{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['poly', 'rbf'], 'degree': [2, 3, 4, 5, 6, 7, 8]}]
```

After extensive testing using trial and error and measuring the results, we discovered the most significant parameters to tune the model are the penalty parameter C, the kernel, the degree, and the gamma. The other parameters are for specific purposes like displaying information or have little impact where the default is ideal in most if not all tests. The penalty parameter tells the SVM optimization how much you want to avoid misclassify each training example. The kernel specifies the type of transformation with different degrees and dimensions. The degree is specific to the polynomial kernel function which measures the degree of the function. The gamma is specific for rbf, polynomial, and sigmoid which defines how much influence a single training example has.

C. Decision Trees

After preprocessing and splitting the data, a GridSearchCV was created and passed a set of parameters (detailed below).

class sklearn.tree. **DecisionTreeClassifier** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*) [source]

Of these, criterion, max_depth, min_samples_split, and min_leaf_nodes were systematically modified and tested, resulting in a best testing accuracy of 93.73% correct. A couple parameters always resulted in better results than their counterparts:

Criterion: gini always performed better than entropy in the tests performed
Splitter: best was chosen as random obviously resulted in either a lack of precision or greater depth.

And a couple were modified and tested with varying results:
Min_samples_split: tested at values 3,5, and 25, initially this was a range as well, but the results were not very different, and the runtime was unacceptably long.
Max_depth: Models were created with max depths that ranged from 3 to 20.

D. Random Forest

The full list of available model parameters for ScikitLearn's implementation of the RandomForest machine learning algorithm sourced from their online documentation at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
It is summarized in the image below of the constructor of the model.

class sklearn.ensemble. **RandomForestClassifier** (*n_estimators='warn', criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None*) [source]

To make a long story short the list contains a lot of the same available tunable parameters that decision trees have and then some ensemble specific tunable parameters. The relevant ensemble specific parameters available to us were: n_estimators, bootstrap. The number of estimators in the ensemble is given by the n_estimators parameter which is of integer type. Whether or not to use bootstrapping is given by the bootstrap which is a boolean.

In the end we decided to only explore the parameters of n_estimators, bootstrap, and max_depth (of the trees in the ensemble). This was done to simplify this complexity of performing this experiment. Of significant note was the use of only max_depth for the trees comprising the ensemble. We felt that restricting this parameter of max depth of the trees alone within a shallow range was enough to provide enough control for the tree estimators rather than additionally playing tree parameters such as minimum gain for splitting, minimum samples, etc. The values we used for each parameter is given in the table below:

n_estimators	25, 50, 100, 150
max_depth	2, 4, 6, 8
bootstrap	True, False

Keep in mind that there are a total of 480 columns for features passed on for trees to split on. At most for a max_depth of 8, each tree can have up to 2^8 nodes or 256 nodes which is more than enough to use all 60 features.

V. Results

A. Neural Networks

The neural network was trained for 150 epochs for different sets of cross validation as well as train/test sets. The table below shows the results for the training and testing sets. The best architecture was a single hidden layer with 50 nodes.

Percent split for test	20%	50%	70%
Training Accuracy	99.61%	99.44%	100%
Testing Accuracy	94.36%	93.67%	91.49%
Stratified CV Accuracy	94.40±1.19%	92.73±2.01%	93.73±2.48%

B. SVM

After extensive testing using the methods and parameters stated prior, we narrowed down the important values down to the following parameter set:

```
[{'C': [100, 1000, 10000], 'gamma': [0.01, 0.001], 'kernel': ['poly', 'rbf'], 'degree': [2, 3, 4]}]
```

This set gives us the following results with their respective training and testing sets. Parameters chosen as best fit for all following SVM tests:

```
{'C': 100, 'degree': 3, 'gamma': 0.01, 'kernel': 'poly'}
```

For a 20% testing ratio:

```
best estimator mean training score   = 0.9996081466330754 +/- 0.00013061786537545714
best estimator mean validation score = 0.9678683385579937 +/- 0.015561908992232719
best estimator test score            = 0.9702194357366771
```

For a 50% testing ratio:

```
best estimator mean training score   = 0.9994427513515088 +/- 0.00027862440348472447
best estimator mean validation score = 0.9605015673981191 +/- 0.018196894016891653
best estimator test score            = 0.9661442006269593
```

For a 70% testing ratio:

```
best estimator mean training score   = 1.0 +/- 0.0
best estimator mean validation score = 0.9540229885057471 +/- 0.02302451957638207
best estimator test score            = 0.961486789072996
```

Overall, we determined that a higher gamma and a polynomial kernel was ideal for the best fit. And as a consequence from polynomial being the best kernel, degree was also tuned with a value of 3 being the best for the polynomial fit. This is unique as rbf is often the more optimal choice in SVM and is the default for the SVC function. Rbf does fit quite well with an accuracy around 2-3% less versus other kernel methods like sigmoid or linear which are at 10+% less.

C. Decision Trees

Percent split for test	20%	50%	70%
Training Accuracy	0.9570	0.9484	0.9564
Testing Accuracy	0.9341	0.9373	0.9305
Std. Dev. 10-fold CV Training Sore	.00145	.00186	.00299
Std. Dev 10-Fold CV Score	.00987	.01428	.02642

D. Random Forest

The following table shows the results for

Train : Test Ratio	Test Score	Mean & Std. Dev 10-Fold CV Score	Mean & Std. Dev. 10-Fold Training Score	Max Depth	N Estimators	Bootstrap?
80:20	0.8981	0.8832 ±0.0196	0.9576 ±0.0029	8	100	False
50:50	0.8715	0.8608 ±0.0279	0.9797 ±0.0035	8	150	False
30:70	0.8540	0.8454 ±0.0315	0.9914 ±0.0028	8	50	False

VI. Discussion

Result Comparison for 20% Test

Algorithm	Best Test Score	Accompanying CV Score	Accompanying Train Score
Random Forest	0.8981	0.8832±0.0196	0.9576±0.0029
Decision Tree	0.9373	0.9347±0.0099	0.9484±0.0015
SVM	**0.9702**	**0.9679±0.0156**	**0.9996±0.0001**
Neural Network	0.9436	0.9440±0.0119	0.9961

Result Comparison for 50% Test

Algorithm	Best Test Score	Accompanying CV Score	Accompanying Train Score
Random Forest	0.8715	0.8608±0.0279	0.9797±0.0035
Decision Tree	0.9373	0.9347±0.0143	0.9484±0.0019

SVM	**0.9661**	**0.9605±0.0182**	**0.9994±0.0003**
Neural Network	0.9436	0.9273±0.0201	0.9961

Result Comparison for 70% Test

Algorithm	Best Test Score	Accompanying CV Score	Accompanying Train Score
Random Forest	0.8540	0.8454±0.0315	0.9914±0.0028
Decision Tree	0.9373	0.9347±0.0264	0.9484±0.0030
SVM	**0.9615**	**0.9540±0.0230**	**1.0000±0.0000**
Neural Network	0.9436	0.9373±0.0248	0.9961

Overall Support Vector Machines did the best in all three train : test split trials. It is difficult to judge whether or not its performance is statistically significant since the top three scores are within their standard deviations of each other. It is entirely possible that perhaps SVM, Neural Networks and Decision Trees are algorithms which perform equally well. One way to test this is to run multiple trials of this experiment with different random seeds to perform a Tukey-Test on the results to measure if the difference of the means and standard deviations were statistically significant. We did not have time to perform such tests.

Neural Networks, SVMs, and decision trees are all very similar to one another and did not have signs of overfitting as their testing results closely matched the validation and training scores. The only model that did suffer from overfitting and was the worst performing algorithm was Random Forest which did show signs of overfitting. Interesting to note however is that the Decision Tree was able to match the performance of the other two best performing algorithms whereas the Random Forest, a tree based ensemble learner, was not. Perhaps if we attempted to additionally use the same tuning parameters that were used in the Decision Tree trials in the Random Forest trials we can match the same results.

System -- Neither -- EI -- IE

KBANN -- 4.62 -- 7.56 -- 8.47
BACKPROP -- 5.29 -- 5.74 -- 10.75
PEBLS -- 6.86 -- 8.18 -- 7.55
PERCEPTRON -- 3.99 -- 16.32 -- 17.41
ID3 -- 8.84 -- 10.58 -- 13.99
COBWEB -- 11.80 -- 15.04 -- 9.46
Near. Neighbor -- 31.11 -- 11.65 -- 9.09

Comparing our results to the results above that the author obtained, given as the error percentage and source from the description of this dataset online at https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Splice-junction+Gene+Sequences).

We were able to meet or exceed them with our models.

VII. Conclusion

In this project, we found that the dataset was fairly easy to classify. Most of our models were consistently scoring above 90%. Compared to the results that the original author had obtained trying certain machine learning algorithms, we were able to meet and or surpass the authors performance. In the future we would like to attempt to do a more in depth analysis of the data by applying sequence visualizations.

Other things we can attempt in the future to analyze why these algorithms performed well on this data set is to apply some existing visualizations specialized on sequence data to find out if perhaps this problem is a class of problems highly separable by SVMs, NNs, and Decision Trees. This however is out of scope for this assignment as it deals in the domain of meta learning. In general, analysis of this problem is difficult because of the high dimensionality of this data and our lack of knowledge of state of the art sequence analysis techniques. Additionally, we do acknowledge the fact that our decision to run three separate trials each differing by train test ratio was arbitrary. This could have been generalized by learning curve data of scores with respect to number of training examples. Doing this would have allowed for another dimension so to speak to compare each algorithm and its efficiency to separate classes and whether or not they have converged or perhaps if more performance can be leveraged by adding more training examples. Along with learning curves, observing the confusion matrices to evaluate the types of errors would have been useful to know to see if certain examples were harder to classify and or if class imbalance was an issue.