Machine Learning Advanced Nanodegree

Capstone Project

Utsav Deep
October 5th, 2018

I. Definition

Domain Background:

The project falls in the category of classification problem. In any classification machine learning problem, the machine learning algorithm tries to assign the given entities with given set of properties or parameters to one of the finite set of given labels or classes. The classical and most common example of classification problem is MNIST digit classification problem. Some other example include: galaxy type classification from given images. Email spam/ham classification, etc.

Article about ***A Random Forest Based Binary Classifier to Predict Bitterness and Sweetness of Chemical Compounds***:

https://www.frontiersin.org/articles/10.3389/fchem.2018.00093/full

Problem Statement:

Given the properties of the molecules, the objective is to build as good a model as possible and as optimally as the given data allows, relate molecular information, to an actual biological response.

I have taken this project from kaggle which is a binary classification problem on predicting probabilities of a biological response of molecules from their chemical properties. The objective is to build as good a model as possible and as optimally as the given data allows, relate molecular information, to an actual biological response.

Dataset and Input

The given data is in the comma separated values (CSV) format. Each row in this data set represents a molecule. The first column contains experimental data describing an actual biological response; the molecule was seen to elicit this response (1), or not (0). The remaining columns represent molecular descriptors (d1 through d1776), these are calculated properties that can capture some of the characteristics of the molecule - for example size, shape, or elemental constitution. The descriptor matrix has been normalized.

Link to the data (This is a kaggle bioresponse competition data):

https://www.kaggle.com/c/bioresponse/download/train.csv

Number of rows for training in the dataset: 3751 rows.

Number of features (columns): There are 1776 different features (molecular descriptors), so dimensionality reduction technique is much needed.

The training set contains 1717 records for molecules not showing any activity; and 2034 records for molecules that are showing the biological response. So we can say that the training set data is almost class balanced.

Evaluation Metrics:

1. ***Logarithmic Loss.***
The evaluation metric specified in the kaggle specific to this project is: Logarithmic Loss.
Log loss is defined as:

$$\log \text{loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y_i}),$$

where N is the number of samples, log is the natural logarithm, $\hat{y_i}$ is the posterior probability that the i^{th} sample elicited a response, and y_i is the ground truth ($y_i = 1$ means the molecule elicited a response, $y_i = 0$ means that it did not).

2. ***Confusion Matrix.***
A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. A typical confusion metric looks like this:

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

This metric is explained in details here.

3. ***Area Under ROC Curve.***
ROC (*Receiver Operating Characteristic*) Curve tells us about how good the model can distinguish between two things (*e.g. If a patient has a disease or no*). Better models can accurately distinguish between the two. Whereas, a poor model will have difficulties in distinguishing between the two. The AUC is the area under the ROC curve. This score gives us a good idea of how well the model performances. This metric is explained in details here.

4. ***Classification Report.***
The classification report visualizer displays the precision, recall, F1, and support scores for the model. This metric is explained in details here.

5. ***Classification Accuracy***
Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Many times classification accuracy gives class biased results. However the training data set I am using is almost class balanced (as explained above). So I have also used classification accuracy as a metric.

II. Analysis

Data Exploration

The given data is in the comma separated values (CSV) format. Each row in this data set represents a molecule. The first column contains experimental data describing an actual biological response; the molecule was seen to elicit this response (1), or not (0). The remaining columns represent molecular descriptors (d1 through d1776), these are calculated properties that can capture some of the characteristics of the molecule - for example size, shape, or elemental constitution. The descriptor matrix has been normalized.

Link to the data (This is a kaggle bioresponse competition data):

https://www.kaggle.com/c/bioresponse/download/train.csv

Number of rows for training in the dataset: 3751 rows.

Number of features (columns): There are 1776 different features (molecular descriptors), so dimensionality reduction technique is much needed.

The training set contains 1717 records for molecules not showing any activity; and 2034 records for molecules that are showing the biological response. So we can say that the training set data is almost class balanced.

Here are some samples of raw data for our understanding:

	Activity	D1	D2	D3	D4	D5	D6	D7	D8	D9	...	D1767	D1768	D1769	D1770	D1771	D1772	D1773	D1774	D1775	D1776
0	1	0.000000	0.497009	0.10	0.0	0.132956	0.678031	0.273166	0.585445	0.743663	...	0	0	0	0	0	0	0	0	0	0
1	1	0.366667	0.606291	0.05	0.0	0.111209	0.803455	0.106105	0.411754	0.836582	...	1	1	1	1	0	1	0	0	1	0
2	1	0.033300	0.480124	0.00	0.0	0.209791	0.610350	0.356453	0.517720	0.679051	...	0	0	0	0	0	0	0	0	0	0
3	1	0.000000	0.538825	0.00	0.5	0.196344	0.724230	0.235606	0.288764	0.805110	...	0	0	0	0	0	0	0	0	0	0
4	0	0.100000	0.517794	0.00	0.0	0.494734	0.781422	0.154361	0.303809	0.812646	...	0	0	0	0	0	0	0	0	0	0

And here is a sample statical description of the raw data:

	Activity	D1	D2	D3	D4	D5	D6	D7	D8	D9	...	D1767	D1768	D1769	D1770	D1771	D1772	D1773	D1774	D1775	D1776
count	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	...	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000	3751.000000
mean	0.542255	0.076948	0.592436	0.068142	0.038990	0.212112	0.686653	0.274713	0.455133	0.749517	...	0.026926	0.014663	0.013863	0.021861	0.015196	0.016796	0.012263	0.011730	0.020261	0.011197
std	0.498278	0.079989	0.105860	0.078414	0.115885	0.102592	0.078702	0.090017	0.162731	0.071702	...	0.161889	0.120215	0.116938	0.146249	0.122348	0.128522	0.110074	0.107683	0.140911	0.105236
min	0.000000	0.000000	0.282128	0.000000	0.000000	0.002630	0.137873	0.006130	0.000000	0.275590	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.033300	0.517811	0.000000	0.000000	0.138118	0.625627	0.207374	0.378062	0.707339	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.066700	0.585989	0.050000	0.000000	0.190926	0.674037	0.277845	0.499942	0.738961	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.100000	0.668395	0.100000	0.000000	0.261726	0.740663	0.335816	0.569962	0.788177	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	0.964381	0.950000	1.000000	1.000000	0.994735	0.790831	0.989870	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Here are some more info about the raw data that I found by running code in python notebook:

1. *No. of missing or NaN values:* ***0***
2. *No. of columns having data with standard deviation < 0.01:* ***7***
3. *No. of columns having absolute correlation (with target column Activity) < 0.05:* ***1258*** *(these columns can be removed as there are more or less noise)*
4. *All data are between 0 and 1. So no outlier handling is required.*

Even though all the data are between 0 and 1 but still their standard deviation are not constant across all columns. Also the data is very high dimensional so dimensionality reduction is required. So some level of data preprocessing required.

Here are top 30 dimensions in decreasing order of their absolute correlation value with the target column:

```
Activity    1.000000 (this is the target column, hence, correlation is 1)
D27          0.472340
D469         0.268601
D217        -0.233387
D182         0.228664
D747         0.226468
D659         0.226049
D87         -0.223081
D146         0.213311
D607         0.210270
D209         0.210154
D660         0.204864
D596         0.204092
D595         0.198737
D187        -0.198590
D979         0.198301
D158        -0.197721
D1036        0.193792
D1087        0.190054
D83          0.184945
D60         -0.180185
D64          0.180094
D10          0.179096
D739         0.178846
D177         0.176242
D1061        0.176234
D959         0.176149
D61         -0.174588
D78          0.172326
D8           0.172226
```

Clearly, the above dimensions are more likely to contribute to the actual classification.

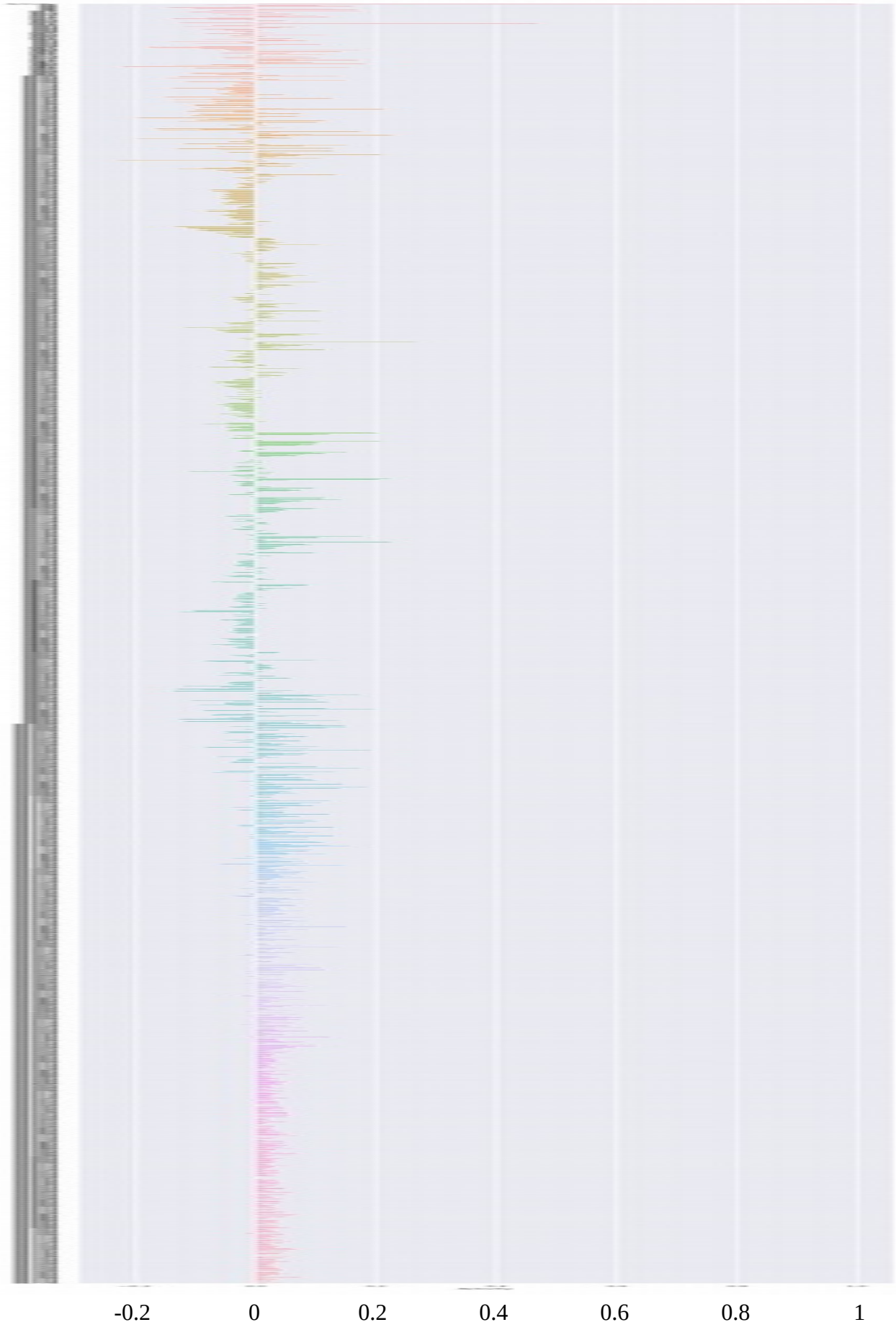
Exploratory Visualizations

In raw data the number of dimensions is huge (1776 dimensions excluding target column). So its quite difficult to do any kind of visualization using all the columns. Size of image becomes very huge even for uni-variate visualizations. However, I have still tried to show 1 uni-variate visualization considering all the dimensions:

Target correlation bar plot. In this plot the y-axis represents the raw data columns ie: Activity and then D1 to D1776 written from top to bottom (its not visible but since column names are continuous values (D1, D2, D3.....) so the y-axis is interpretable). The x-axis is the correlation value of the column with respect to the target column (Activity). Since it wasn't easy to see, I have written the x-axis values below the plotted image.

From the below plot we can safely agree that more than three-forth of the dimensions have less than 0.05 absolute correlation value with target column. These columns can be safely removed from the

data frame as these are possible noise. Also reducing the dimensions saves a lot of traing and testing time.



Algorithms and Techniques

Following are the algorithms that I have considered and used it for predicting the target class for this binary classification problem:

1. **Logistic Regression**:
   It is the go-to method for binary classification problems (problems with two class values). So its was worth a try for this project which is also a binary classification problem. In this algorithm the prediction for the output is transformed using a non-linear function called the logistic function.
   The logistic function looks like a big S and will transform any value into the range 0 to 1. This is useful because we can apply a rule to the output of the logistic function to snap values to 0 and 1 (e.g. if less than 0.5 then output 1) and predict a class value.
2. **Support Vector Machine (SVM):**
   SVM is considered as one of the most powerful out-of-the-box classifiers and worth trying especially in binary classification problems.
   In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions, one can visualize this as a line and let's assume that all of our input points can be completely separated by this line. The SVM learning algorithm finds the coefficients that results in the best separation of the classes by the hyperplane.
   I have taken SVM as benchmark model also but for making benchmark predictions I haven't applied any dimensionality reduction and data standardisation, but in improved one I did.
3. **Neural Network:**
   In any type of machine learning problem, whether its classification, regression, reinforcement or unsupervised, neural networks are expected to outperform other algorithms (obviously good enough NN models need to be selected specific to the target problem). So its was a worth to try out neural network.
   Here is the details of the neural network model I have used for this problem:
   Library used: keras
   Number of input nodes: 518 (equal to the number of dimensions in the processed dataframe)
   Number of hidden layers: 2 with 20 nodes in each layer
   Hidden layer 1 activation function: relu (rectified linear unit)
   Hidden layer 2 activation function: softmax (rectified linear unit)
   Output layer activation function: sigmoid
   Loss: binary_crossentropy
   Optimizer: adam
   Output threshold: 0.5
   Epoches: 500
4. **Random Forest:**
   Random forest is capable of both regression and classification. In this problem the data is very high dimensional even after applying dimensionality reduction. Due to this random forest was a worth try as it can handle a large number of features, and it's helpful for estimating which of the variables are important in the underlying data being modeled. It belongs to a larger class of machine learning algorithms called ensemble methods.

Ensemble learning involves the combination of several models to solve a single prediction problem. It works by generating multiple classifiers/models which learn and make predictions independently. Those predictions are then combined into a single (mega) prediction that should be as good or better than the prediction made by any one classifer.
Random forest model gave the best result among the algorithms I have tried. The parameters that I used are: `n_estimators=100, max_depth=1000`

Before training or making prediction with all of the above models. Dataframe was standardized and dimensionality reduction technique was applied.

Benchmark

I have chosen Support Vector Machine (SVM) model as benchmark model. I am also using SVM as one of the improved models but the difference is that for benchmark model, I am not applying any data standardizaion or dimensionality reduction techniques. I could have taken Logistic Regression or Naive Bayes as benchmarks but since I wanted to make this problem more challenging, I took SVM as benchmark which is well known to give very good results especially in case of binary classificaion problems. I will consider all the metrics discussed in the overview section of the report and outperform each metrics with one or more of the ML models discussed above. Here are the values of the performance metrics for the benchmark solution:

---

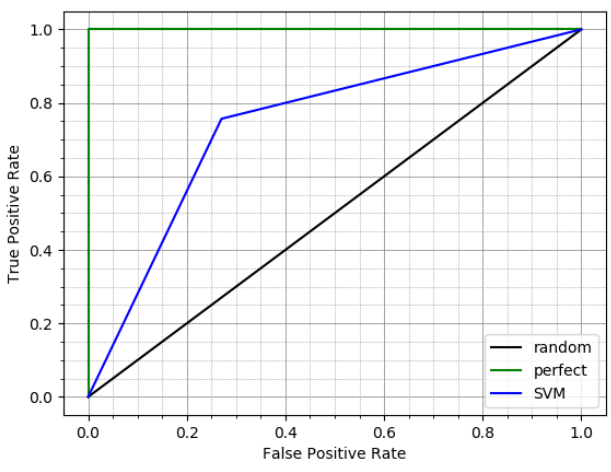
confusion_matrix:
[[251 93]
 [99 308]]

classification_report:

	precision	recall	f1-score	support
0	0.72	0.73	0.72	344
1	0.77	0.76	0.76	407
micro avg	0.74	0.74	0.74	751
macro avg	0.74	0.74	0.74	751
weighted avg	0.74	0.74	0.74	751

accuracy_score: 0.744
roc_auc_score: 0.743
log_loss: 8.830

ROC Curve:



These benchmark performance metrics are already better than many of the top submissions in Kaggle for this problem.
For obtaining the above results, here are the steps I followed:

1. converted csv data from train.csv into raw dataframe
2. split the above raw dataframe into custom train and test data frame
3. trained the train_dataframe with SVM model using sklearn
4. obtained the predicted values from the trained model
5. calculated the above metrics using sklearn.metrics

III. Methodology

Data Preprocessing

The input data for this project is pretty much clean contrary to any typical machine learning challenge. As explained in data exploration sub-section of Analysis section, the data is free from NaN or missing values, there are no catagorical values (except in target column which must have it) and free from outliers (as all the values are between 0 and 1). However some of the columns contains almost constant values and the data is not scaled properly. Also the data is very high dimensional, so dimensionality reduction should also be done. So here are the steps in data preprocessing for this problem:

1. **Removal of columns having almost constant data:** The columns having standard deviation less than 0.01 were removed as these are almost constant data and will have negligible contribution in predicting the target class.
2. **Dimensionality Reduction:** The columns which have correlation value less than 0.05 with target column (‘Activity’) are removed from the dataframe. This way of dimensionality reduction gave slightly better performance metrics than with PCA(0.95) in almost all the models.
3. **Data standardisation and columns sorting:** The dataframe is standardised such that its mean and std of data in every column become 0 and 1 respectively. Finally feature columns are sorted in descending order according to their absolute correlation value with respect to the target column.

Implementation

The project is implemented in a well structured way. Here are the key points:

1. All the common properties are kept in src/commons/constants.py. This is done to minimise the use of hard coded variables.
2. All the models are kept in the module: src/models and they all follow the same template so that its easy to switch between the models
3. Dataframes are saved as .pickle file after preprocessing. This is done to avoid necessarily running the preprocess.py every time data visualization or model training or model evaluation is run.
4. Exploratory visualization plots are saved as a file rather than displaying it on every run.

5. The models are saved in external file after training. This is done to avoid running train_models everytime model evaluation is run or prediction of test data is run.
6. Comments for important methods are written in detail. Proper naming convention and PEP-8 style guideline is followed.
7. Execution report is generated in the notebook. Its kept inside notebooks folder.
8. The project should be run in the following sequence esp., while running it first time:
    - src/data/preprocess.py
    - src/visualization/exploratory.py
    - src/models/train_models.py
    - src/evaluation/model_performance.py

Here is the project directory tree that clearly explains the structure of the project:

```
.
├── data
│   ├── external
│   ├── interim
│   ├── processed
│   │   └── processed.pickle
│   └── raw
│       ├── raw.pickle
│       └── train.csv
├── environment.yml
├── models
│   ├── benchmark
│   │   └── SVM.model
│   ├── KerasNN.h5
│   ├── KerasNN.yaml
│   ├── LogisticRegression.model
│   ├── RandomForest.model
│   └── SVM.model
├── notebooks
│   └── execution_report.ipynb
├── README.md
├── reports
│   ├── figures
│   │   ├── evaluation
│   │   │   ├── benchmark_roc_curve.png
│   │   │   └── roc_curves.png
│   │   ├── processed_target_corr_plot.png
│   │   └── raw_target_corr_plot.png
│   └── model_performance.txt
└── src
    ├── commons
    │   ├── constants.py
    │   ├── __init__.py
```

```
├── data
│   ├── __init__.py
│   ├── preprocess.py
├── evaluation
│   ├── __init__.py
│   ├── model_performance.py
├── __init__.py
├── models
│   ├── __init__.py
│   ├── keras_nn.py
│   ├── logistic_regression.py
│   ├── random_forest.py
│   ├── svm.py
│   └── train_models.py
└── visualization
    ├── exploratory.py
    ├── __init__.py
```

Model Implementations:

- **random_forest.py**

```python
import pickle
import sys
from sklearn.ensemble import RandomForestClassifier
sys.path.append('src')
from src.data.preprocess import get_featues, get_label
class RandomForestModel(object):
    def __init__(self):
        self.clf = RandomForestClassifier(n_estimators=100, max_depth=1000)
        self.name = 'RandomForest'
    def get_params(self):
        return self.clf.get_params()
    def train(self, dframe):
        X = get_featues(dframe)
        y = get_label(dframe)
        self.clf.fit(X, y)
    def predict(self, X):
        y_pred = self.clf.predict(X)
        return y_pred
    def predict_proba(self, X):
        y_pred = self.clf.predict_proba(X)
        return y_pred
    def save(self, fname):
        with open(fname, 'wb') as ofile:
            pickle.dump(self.clf, ofile, pickle.HIGHEST_PROTOCOL)
    def load(self, fname):
        with open(fname, 'rb') as ifile:
            self.clf = pickle.load(ifile)
```

- **keras_nn.py**

```python
import sys
```

```
from keras.layers import Dense
from keras.models import Sequential
from keras.models import model_from_yaml
sys.path.append('src')
from src.data.preprocess import get_featues, get_label
class KerasNN(object):
    def __init__(self):
        self.clf = Sequential()
        self.clf.add(Dense(20, input_dim=518, activation='relu'))
        self.clf.add(Dense(20, activation='softmax'))
        self.clf.add(Dense(1, activation='sigmoid'))
        self.clf.compile(loss='binary_crossentropy', optimizer='adam')
        self.name = 'KerasNN'
    def train(self, train, validation):
        X = get_featues(train)
        y = get_label(train)
        xv = get_featues(validation)
        yv = get_label(validation)
        self.clf.fit(X, y, epochs=500, verbose=0, validation_data=(xv, yv))
    def predict(self, X):
        y_pred = self.clf.predict(X)
        y_pred = (y_pred > 0.5) * 1
        return y_pred
    def predict_proba(self, X):
        # this is done to overcome divide by zero error while calculating
log_loss metric
        y_pred = (self.clf.predict_proba(X)+0.00001)*0.9999
        return y_pred
    def save(self, fname):
        model_yaml = self.clf.to_yaml()
        yaml_file_name = fname + '.yaml'
        with open(yaml_file_name, "w+") as yaml_file:
            yaml_file.write(model_yaml)
            # serialize weights to HDF5
            weights_file_name = fname + '.h5'
            self.clf.save_weights(weights_file_name)
    def load(self, fname):
        yaml_file_name = fname + '.yaml'
        with open(yaml_file_name, 'r') as yaml_file:
            loaded_model_yaml = yaml_file.read()
            self.clf = model_from_yaml(loaded_model_yaml)
            # load weights into new model
            weights_file_name = fname + '.h5'
            self.clf.load_weights(weights_file_name)
```

The other two models svm and logistic_regression also follow the above class template (ie have same method signatures) but are initialized with default parameters.

Refinement

Initial models: All the models were trained without preprocessing the raw dataframe (the models wont break as the raw data for this problem was already quite clean). For random forest model I initially used n_estimators=20, max_depth=100.
For KerasNN model I initially used input layer with 1776 nodes; 2 hidden layers with 5 nodes each, and activation function relu; 1 output node with sigmoid activation function and default values for other parameters.

For SVM and LogisticRegression models, I used default parameters.
Performance metrics were worse than the benchmark model for all the models (except SVM which is, in this case, effectively same as benchmark model).

Process of refinement: I tried fine tuning of various hyper parameters like changing the number of estimators and max depth in case of random forest, changing the number of nodes, activation function, etc in case of keras neural network model, etc. I also tried dimensionality reduction with PCA but it gave slightly worse result than my custom dimensionality reduction technique.

Final model: All the models (obviously except benchmark model) used preprocessed data for both training and testing.
For SVM and LogisticRegression models, I used default parameters even in final model.
For random forest model I used n_estimators=100, max_depth=1000.
For KerasNN model used following hyper-parameters:

- *Number of input nodes: 518 (equal to the number of dimensions in the processed dataframe)*
- *Number of hidden layers: 2 with 20 nodes in each layer*
- *Hidden layer 1 activation function: relu (rectified linear unit)*
- *Hidden layer 2 activation function: softmax (rectified linear unit)*
- *Output layer activation function: sigmoid*
- *Loss: binary_crossentropy*
- *Optimizer: adam*
- *Output threshold: 0.5*
- *Epoches: 500*

IV. Results

Model Evaluation and Validation

Here are the detailed results for all the considered models in terms of various performance metrics:

```
Benchmark model performance:
```

```
Performance of SVM
--------------------------------------------------------------------------------
confusion_matrix:
[[251  93]
 [ 99 308]]
classification_report:
              precision    recall  f1-score   support

           0       0.72      0.73      0.72       344
           1       0.77      0.76      0.76       407

   micro avg       0.74      0.74      0.74       751
   macro avg       0.74      0.74      0.74       751
weighted avg       0.74      0.74      0.74       751
accuracy_score:    0.744
roc_auc_score: 0.743
log_loss:  0.524
```

```
Improved models performances:
```

```
Performance of RandomForest
--------------------------------------------------------------------------------
```

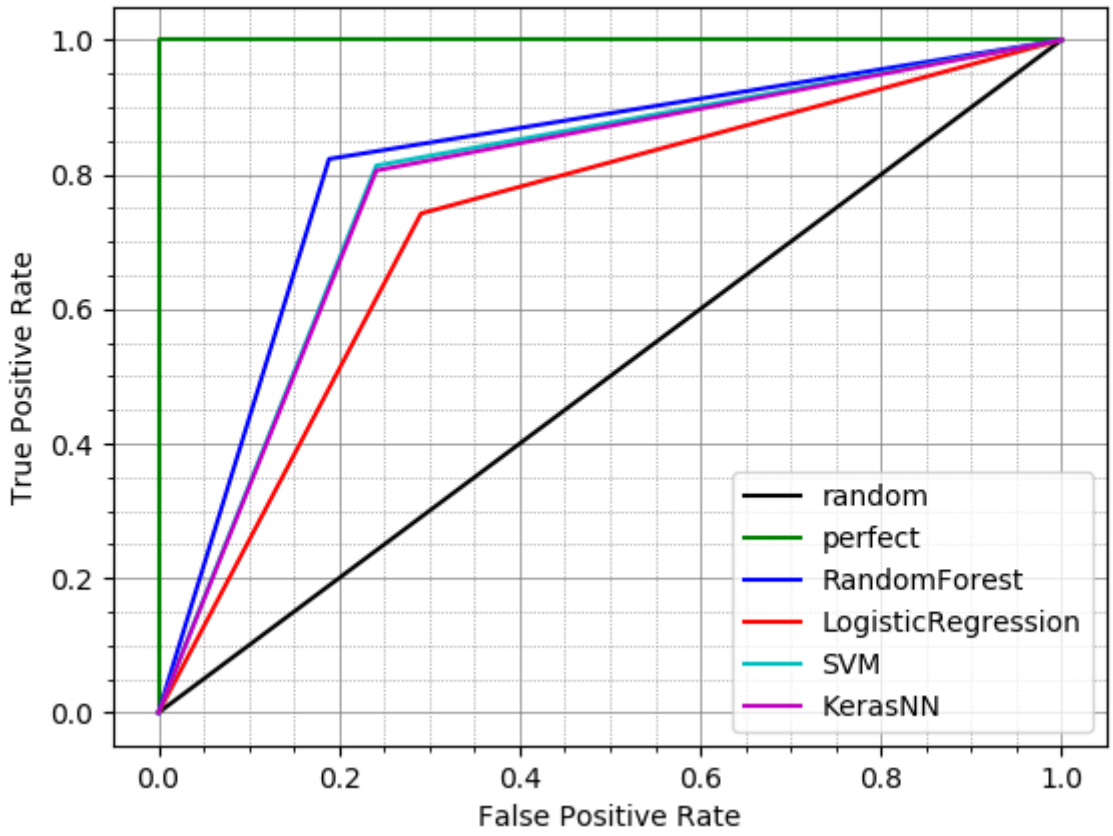
```
confusion_matrix:
[[279  65]
 [ 72 335]]
classification_report:
              precision    recall  f1-score   support
           0       0.79      0.81      0.80       344
           1       0.84      0.82      0.83       407
   micro avg       0.82      0.82      0.82       751
   macro avg       0.82      0.82      0.82       751
weighted avg       0.82      0.82      0.82       751
accuracy_score:    0.818
roc_auc_score: 0.817
log_loss:  0.438
```

```
Performance of LogisticRegression
--------------------------------------------------------------------------------
confusion_matrix:
[[244 100]
 [105 302]]
classification_report:
              precision    recall  f1-score   support
           0       0.70      0.71      0.70       344
           1       0.75      0.74      0.75       407
   micro avg       0.73      0.73      0.73       751
   macro avg       0.73      0.73      0.73       751
weighted avg       0.73      0.73      0.73       751
accuracy_score:    0.727
roc_auc_score: 0.726
log_loss:  0.703
```

```
Performance of SVM
--------------------------------------------------------------------------------
confusion_matrix:
[[261  83]
 [ 76 331]]
classification_report:
              precision    recall  f1-score   support
           0       0.77      0.76      0.77       344
           1       0.80      0.81      0.81       407
   micro avg       0.79      0.79      0.79       751
   macro avg       0.79      0.79      0.79       751
weighted avg       0.79      0.79      0.79       751
accuracy_score:    0.788
roc_auc_score: 0.786
log_loss:  0.461
```

```
Performance of KerasNN
--------------------------------------------------------------------------------
confusion_matrix:
[[261  83]
 [ 79 328]]
classification_report:
              precision    recall  f1-score   support
           0       0.77      0.76      0.76       344
           1       0.80      0.81      0.80       407
   micro avg       0.78      0.78      0.78       751
   macro avg       0.78      0.78      0.78       751
weighted avg       0.78      0.78      0.78       751
accuracy_score:    0.784
roc_auc_score: 0.782
log_loss:  0.844
```

ROC Curves for above Models:



Clearly, RandomForest model beats all other models in terms of all the metrics that were considered. Hence I will select this RandomForest model as the solution model for this problem. The hyper parameters, (n_estimators=100, max_depth=1000), were properly fine tuned before reaching to this solution model. Hyper-parameters of other models were also fine-tuned but that wasn't sufficient enought to make it the best model.

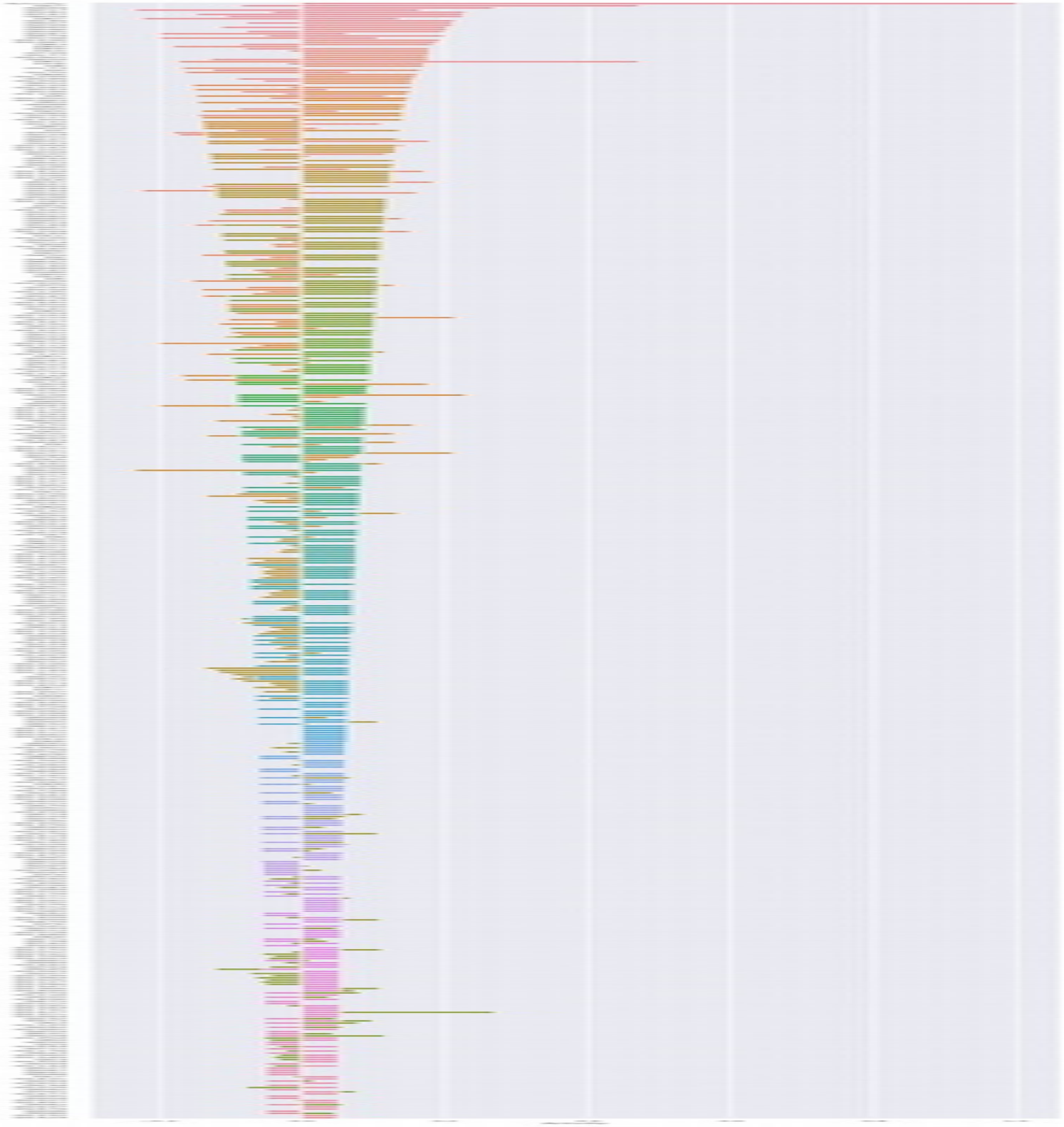
Justification

The difference in perfomance metrics and ROC graphs of benchmark SVM model and RandomForest model clearly shows that RandomForest model beats benchmark model in all the considered performance metrics.
RandomForest model works by generating multiple classifiers/models which learn and make predictions independently. Those predictions are then combined into a single (mega) prediction that should be as good or better than the prediction made by any one classifer. So it is expected to give better results if we increase the number of estimators and max depth. However after some extent of fine-tuning, like every model, this model also gets saturated in terms of performance metrcs.

This model successfully achieved 80-85% success rate in almost all the considered metrics. Also the log loss value of `0.438` is among the top submissions in kaggle.

V. Conclusion

Free-Form Visualization



The above visualization show the feature columns that were used to make predictions in this project. The plot shows the first 518 columns from highest to lowest that contribute the most in the molecular activity prediction.

Reflection

First I read some common and elegent ways to structure any machine learning project and followed one of the project structures. Then I learnt several machine learning algorithms and chose some of them that were best suited for binary classification problems as the project I dis is a binary classificaiton problem. After that I listed down all the steps upto most basic level that I have to do for completing this project and created classes and methods with empty bodies according to the reqired actions that I had to do. Then I installed all the ML libraries in anaconda virtual environment. After that I downloaded the train.csv data of this project from kaggle and started started exploring the data using jupyter notebook. After getting good enough idea about the raw data, I started implementing the functionalities in the project that I have created with skeleton methods. Then I integrated all the parts of the project together and started doing fine tunings of the models. I also refactored the project from time to time to make it better in terms of readablity and performance.

The intersting aspect about the problem was unlike other typical kaggle challenges, the training data for this challenge was quite clean. So I didn't have to care about filling NaN values, outliers and categorical values.

The challenging aspect about the problem was the data set for this problem had a lot of dimensions (or feature columns), making any kind of visualizaion very difficult.

The final model and the solution successfully surpassed the benchmark model by significant margin with respect to all the metrics that were considered. This was inspite that the benchmark model already had very good performance metrics with around 70-75% success rate.

I will be using this solution project as a general setting to classificaion problems, esp, binary classification problems.

Improvement

I think the neural network model underperformed a lot and with better fine tuning it can outperform the RandomForest model that, I have created, by good enough margin. There were many other algorithms that exist and possibly it can possibly outperform the models that I used in the project. However I yet have to explore those algorithms in detail before I can correcly use them.

My solution model's log loss value is only 0.438 but the best solution submitted in Kaggle has a log loss value of 0.37355. So, clearly there can be better solutions than mine and my solution can work as a benchmark for those models.