Report on Drug Discovery

ID: math.random12 **Accuracy:** 0.75 **Rank:** 6

Approach Followed to solve the problem:

There are several steps performed before classifying compounds with unknown labels. Those are broadly categorized into Data Analysis, Feature extraction, Model selection and Hyperparameter tuning, Cross validation and Visualization,Classifying unknown documents in the end. However, the ***majority*** of effort went into ***regularization*** because the data was very imbalanced. Details of each step are discussed in the following sections.

1. Data Analysis

If we look at the data set the target label is binary and it contains a sequence of numbers representing the properties of that compound. The data set is imbalanced as mentioned in the problem statement there are **722 inactive** and **78 active** compounds. This imbalance is an indication that either we have to **under-sample** or **over-sample** our data to make unbiased predictions. I choose to go with **over-sampling** because undersampling could gerid of some records containing useful information. The sampling method used is **random and with replacement**.

2. Feature Extraction

As the Data is already cleaned. I just converted it to a binary vector. Each place in the vector represents if the feature is present in the compound or not. Additionally, some features are removed based on some criteria which are discussed in the following section.

3. Model selection and Hypermeter tuning

The major part of efforts in solving this problem went into **regularization** and avoiding **overfitting.** I choose to go with Decision Tree and Neural Networks. Because, with an imbalanced data set, a probabilistic model like Naive Bayes classifier maynot be the best one to choose. Because, the **final prediction** is always dependent on the **prior probability**. In this case **not-active compounds** will have a **higher weightage** over active ones. Even if we apply oversampling, based on certain features, the probabilistic model could make a biased prediction.

3.1 Decision Tree

As the data is less, the decision tree started to **overfit on the training** data as soon as I started training it. Which performed very badly on the testing data set. To avoid overfitting I tried **2 different methods. Pre-pruning** and **Post-pruning. Pre-pruning** however, was **not able** to produce **good results**. As it is hard to understand optimal **depth** of the tree and **minimum records to split** at any level by just looking at the data.

I found **Post-pruning** promising as it prunes the tree and then determines the accuracy and further prunes it. I was able to get a **boost of 5%** in the **F1-Score** after using post pruning in miner2. I used **cost-complexity-pruning** method. In this method we assign a score to every tree. The score is defined as $Tree\ Score = Misclassification + \alpha * (Number\ of\ terminal\ nodes)$**.** The idea is to **reduce the number of leaves** and **miss classification error** while increasing the alpha. Finally the tree alpha where the tree has **higher training and testing accuracy** will be chosen.

The following graph **[Fig-1]** on the left side is the demonstration of the **F1-score**. Graph on the right is **depth vs number of nodes**. In both of the graphs **alpha value is on the Horizontal axis** .

3.2 Neural network

Neural networks worked promisingly in this task. And out **performed accuracy of decision trees** by **6%** based on miner2.

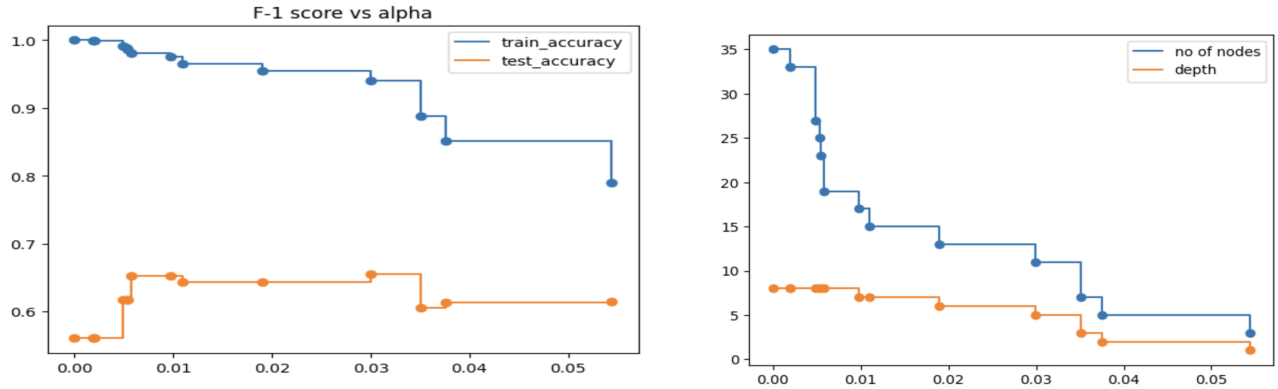


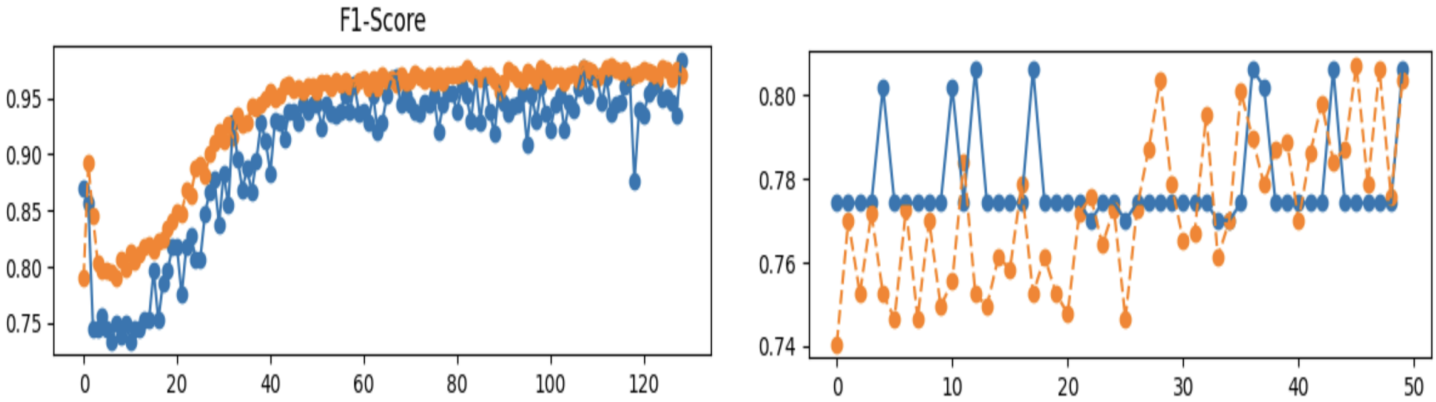
Figure - 1

The majority of effort went into choosing the best size neural network with respective regularization hypermeter. As soon as I started training ***without regularization*** the NN started overfitting the data data string from 2nd iteration. After some research it was apparent to use regularization parameters. I had 3 options: ***L1, L2 regularization*** and ***Dropout*** method. ***L1 and L2 penalizes weights*** while drop out ***omits certain neurons.*** L1 could make a weight 0 while ***L2 is extremely slow*** in pushing weight ***towards 0***. I **did not use the feature subse**t. The thought process is, I *wanted the NN to get optimal weights for each on the input*. In other words, I rather wanted the *NN to decide which feature to omit/consider-less using loss function*.
As I wanted the **weights to be 0** for irrelevant features so, I used **L1 regularization.**

Choosing the **right λ for L1** regularization was another challenging task. To come up with a value It tried **bigger values first**. They did **not** let the model **fit to the data**. Then I tried **smaller values** which led to **overfitting**. Sort of like binary search I used intermediate values to come up with the value that worked.

Initially, I used ***Binary Cross Entropy*** as the loss function but the issue was it was not able to produce good results. As it does not take imbalance into account. Also, gradient descent updates the weight for prediction towards the confident class. Because, the ***majority class dominates*** the ***loss function***. **Binary Focal Cross Entropy** however, was a better choice because it weights class prediction and forces the model to learn parameters towards the hard to classify class. The following is the distinction between model ***F1-Score*** during training and testing using *Binary Cross Entropy* on the **Right** and *Binary **Focal** Cross Entropy* on the **Left.** As it is clear that Focal Cross Entropy converges a lot better than the counterpart Binary cross entropy.

Testing: **Training:**



From the graph it is apparent to choose Focal cross entropy as the final loss for our model. This model is chosen for final prediction on the unlabeled data and has a **75% accuracy** according to **Miner2**.