Project Report

Our model aims at predicting whether a person is affected by a disease or not using Edge Intelligence with minimum latency. We are using a labeled dataset consisting of various symptoms of diseases. Dataset is divided into subsets of training dataset and testing dataset. Using training dataset, our model finds patterns and using that knowledge history it finds outcomes from test dataset.

We have tried various machine learning techniques that fit for binary classification and compare among themselves to find the best fit to the dataset. For building each suitable model with optimal hyper parameters setting, grid search is performed within k-fold cross validation. The suitable models are then stored for later comparison.

Machine Learning Techniques

1. Logistic Regression:

Equation for logistic regression is

The hyperparameter C is a regularization parameter that controls how closely the model fits to the training data. We have tuned this C parameter to find out the optimal value for our dataset. Both training speed and prediction speed is fast for this algorithm, but it’s performance is poor compared to other algorithms.

lr = LogisticRegression()

parameters = {

'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],

}

cv = GridSearchCV(lr, parameters, cv=5)

cv.fit(tr\_features, tr\_labels)

2. Support Vector Machine (SVM)

C and kernel hyperparameters are tuned to find out the optimal value for our dataset. Training speed is very slow for this algorithm. It gives very good performance when the dataset used is small. Its performance degrades when the number of rows in the dataset increases.

svc = SVC()

parameters = {

'kernel': ['linear', 'rbf'],

'C': [0.1, 1, 10]

}

cv = GridSearchCV(svc, parameters, cv=5)

cv.fit(tr\_features, tr\_labels)

3. Random Forest:

Parameters n\_estimators and max\_depth are tuned. N\_estimators represent how many decision trees are built and max\_depth represents maximum depth of each tree.

rf = RandomForestClassifier()

parameters = {

'n\_estimators': [5, 50, 250],

'max\_depth': [2, 4, 8, 16, 32, None]

}

4. Boosting:

Parameters n\_estimators, max\_depth and learning rate are tuned.

gb = GradientBoostingClassifier()

parameters = {

'n\_estimators': [5, 50, 250, 500],

'max\_depth': [1, 3, 5, 7, 9],

'learning\_rate': [0.01, 0.1, 1, 10, 100]

}

For implementing Machine Learning techniques we have used sklearn package.

Deep Learning Technique

DeepNet Model:

def \_\_init\_\_(self, shape\_x, shape\_y, opt, l, neurons, mname):

self.model=keras.Sequential([keras.layers.Flatten(input\_shape=(shape\_x,shape\_y))])

for n in neurons:

self.model.add(keras.layers.Dense(n, activation='relu'))

self.model.add(keras.layers.Dense(shape\_y, activation='sigmoid'))

self.model.compile(optimizer=opt, loss=l, metrics=['accuracy'])

self.mname=mname

The neurons in input and the hidden layers are considered as per the features in the following dataset as the accuracy of the model depends on the number of features provided and as well as the number of neurons being considered. For example, for the Diabetes dataset we have used the following,

dlmod=dn.DeepNet(8, 1,'adam','binary\_crossentropy',[12,8],"Diabetes")

We have considered 80 percent of the dataset as training dataset and rest as test dataset.

Number of epoch is considered as 250 and batch size as 10.

Epoch means one pass through all of the rows in the training dataset.

Batch Size means one or more samples considered by the model before weights are updated.

For optimizer we have taken ‘adam’ and loss function as ‘binary\_crossentrophy’.

The reason to choose ‘adam’ as an optimizer as this is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems. and for loss function ‘binary\_crossentrophy’ is selected for its binary classification as the target values are in binary 0 and 1 with crossentrophy as loss argument.

Comparisons

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training Speed | Prediction Speed | Performance | Performance with limited data |
| Logistic Regression | Fast | Fast | Lower | Higher |
| SVM | Slow | Moderate | Medium | Higher |
| Random Forest | Moderate | Moderate | Medium | Lower |
| Boosting | Slow | Fast | Medium | Lower |
| DeepNet | Slow | Moderate | High | Moderate |

Performance of machine learning algorithms deteriorates with increasing size of datasets,while deep learning algorithms can handle large datasets well. Our deepnet model performs well when the number of features in the dataset increases.

The DeepNet model predicts the results for the Heart dataset in 0.060 secs for the testing data and in 0.125secs for the Diabetes dataset’s testing data.