LAB 4

MINING ADVISOR-ADVISEE RELATIONSHIPS

May 22, 2020

Name:Chenyu Yang
Student ID: 517030910386
Class: F1703015

Contents

1 INTRODUCTION

This Lab focuses on learning basic machine learning methods and implementing them on a specific topic, to find advisor-advisee relationships in academic heterogeneous networks. In this Lab, you will learn some machine learning tools and realize your own model based on Python and Sklearn. Moreover, you will use Keras and Tensorflow to build a deep-learning model and compare the performance between deep learning and traditional machine learning methods.

2 BACKGROUND

2.1 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras has the following advantages:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

2.2 Dataset

In this experiment, we have the known (GroundTruth) Advisor-advisee Relationships (AARs) and common coauthor relationships, which are obtained through calculating the probability of being an AAR respectively. The authors are represented by eight 4-bit code.

Features have been extracted in the perspective of relationship before, in and outside one cooperation. For instance, if we have known A cooperated with B in paper publication from 2008, then the more paper A has before 2008 than B, the more likely that A is Advisor and B is Advisee in their cooperation. We have 22 ordered features ranked through Mutual Information Correlations, method of feature engineering (realized in Python sklearn).

3 PROCEDURE

3.1 Deep Neural Network

Firstly, we test a deep neural network. Deep neural network(DNN) is based on gradient update. The forward calculation tries to get a label $\hat{y}$ according to features X while the

backword propagation update parameters in model based on the error between $\hat{y}$ and ground truth y.

In this experiment, we use a simple 2-layer perceptron with a nonlinear activation 'sigmoid'. We use a softmax function to get the final probability for classification.

```
model = Sequential()
model.add(Dense(5, input_shape = (shape_in,))) #input scale
model.add(Activation('sigmoid'))
model.add(Dense(2)) #output scale
model.add(Activation('softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
    ])
model.fit(X_train, y_train, nb_epoch=5, batch_size=5, verbose=1)
```

3.2 Support Vector Machine

Support vector machine (SVM) is a two classification model. Its basic model is a linear classifier with the largest interval defined in the feature space, which makes it different from the perceptron. SVM also includes kernel techniques, which makes it a non-linear classifier in essence. The learning strategy of SVM is to maximize the interval, which can be formalized as a problem of solving convex quadratic programming, and also equivalent to the problem of minimizing the regularized hinge loss function. The learning algorithm of SVM is the optimization algorithm for convex quadratic programming.

We scaled the dataset to make the model more convergent and tried different kernels for SVM.

```
scaler=StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
model = SVC(kernel = "rbf")
model.fit(X_train, y_train)
```

3.3 Decision Tree

Decision tree is a decision analysis method which can get the probability that the expected value of NPV is greater than or equal to zero, evaluate the project risk and judge its feasibility by constructing decision tree on the basis of knowing the probability of occurrence of various situations.

Decision tree is a very common classification method. It is a kind of regulatory learning. The so-called regulatory learning is to give a group of samples, each of which has a set of attributes and a category. These categories are determined in advance. Then a classifier can be obtained by learning, which can give correct classification to the new objects. Such machine learning is called supervised learning.

```
model = tree.DecisionTreeClassifier(min_samples_split = 40)
model.fit(X_train, y_train)
```

4 RESULT

For the DNN, we use ***Adam*** as the optimizer and ***cross entropy*** as the loss fuction. We trained the model for 5 epoches and the batch size is set to 5.

For SVM, we adapted 2 different kernels: ***rbf*** and ***linear***. And before fitting the model we scale the dataset using ***StandardScaler***.

For decision tree, we set the ***min_samples_split*** to 40. This model doesn't need the dataset been scaled.

Model	Loss	Accuracy	Time
DNN	16.88	94.57	42.18
SVM(linear)	-	94.12	4.26
SVM(rbf)	-	94.13	2.44
DTree	-	94.52	0.39

Table 1: Results for different models

As the table shows, for this lab, DNN and Decision Tree performs better than SVM. but DNN takes long time to train. Decision tree is the fastest model of them all.

5 CONCLUSION

In this lab, we used different models to predict the adviser-advisee relationships, which makes me know more about machine learning.

6 COMPLETE CODE

https://github.com/Achronferry/SJTU-EE447-Lab/tree/master