Machine Learning Algorithms -Part2:

Tasks:

1. Build different models to predict whether a tweet is positive, neutral or negative, and if it is negative, give the type of the reason of it being negative.
2. Base on the run time and accuracy of different model, choose the best model for this particular task.
3. Avoid overfitting.

Model:

Naïve Bayes:

Naïve Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

Neural Network:

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well.

Technique:

We used scikit-learn package(sklearn.neural_network) and Spark Multilayer perceptron classifier to compare the result of neural network.

sklearn.neural_network package:

```
# Models we will use
logistic = linear_model.LogisticRegression()
rbm = BernoulliRBM(random_state=0, verbose=True)

classifier = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])

# Hyper-parameters. These were set by cross-validation,
# using a GridSearchCV. Here we are not performing cross-validation to
# save time.
rbm.learning_rate = 0.06
rbm.n_iter = 20
# More components tend to give better prediction performance, but larger
# fitting time
rbm.n_components = 100
logistic.C = 6000.0

# Training RBM-Logistic Pipeline
classifier.fit(X_train, Y_train)

# Training Logistic regression
logistic_classifier = linear_model.LogisticRegression(C=100.0)
logistic_classifier.fit(X_train, Y_train)

########################## Testing ############################

print()
print("Logistic regression using RBM features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        classifier.predict(X_test))))

print("Neural Network using raw pixel features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        logistic_classifier.predict(X_test))))
```

Spark Multilayer perceptron classifier:

```
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.sql.Row

// Load training data
val data = MLUtils.loadLibSVMFile(sc, "hdfs://cshadoop1/user/gxy140830/TweetsAll.csv").toDF()
// Split the data into train and test
val splits = data.randomSplit(Array(0.6, 0.4), seed = 1234L)
val train = splits(0)
val test = splits(1)
// specify layers for the neural network:
// input layer of size 4 (features), two intermediate of size 5 and 4 and output of size 3 (classes)
val layers = Array[Int](4, 5, 4, 3)
// create the trainer and set its parameters
val trainer = new MultilayerPerceptronClassifier()
  .setLayers(layers)
  .setBlockSize(128)
  .setSeed(1234L)
  .setMaxIter(100)
// train the model
val model = trainer.fit(train)
// compute precision on the test set
val result = model.transform(test)
val predictionAndLabels = result.select("prediction", "label")
val evaluator = new MulticlassClassificationEvaluator()
  .setMetricName("precision")
println("Precision:" + evaluator.evaluate(predictionAndLabels))
```

We used naiveBayesClassifier package and MLlib Naïve Bayes to compare the result of Naïve Bayes.

naiveBayesClassifier package:

```
from naiveBayesClassifier import tokenizer
from naiveBayesClassifier.trainer import Trainer
from naiveBayesClassifier.classifier import Classifier
```

Accuracy:

Naïve Bayes:

If we only need to predict three classes(Positive, Negative, and Neutral), it could have the accuracy of 0.794696969697.

```
cometnet-10-21-1-19:Big_Data_Project Helicopter$ ./NB_3cls.py
14640
Training...
Testing...
0.794696969697
```

If we only need to predict all twelve classes (including all classification of different reasons of negative), it could have the accuracy of 0.381060606061, which is relatively good because some of the different reasons are very alike.

```
cometnet-10-21-1-19:Big_Data_Project Helicopter$ ./NBAir.py
14640
Training...
Testing...
0.381060606061
```

Neural Network:

Neural network could have the accuracy of 86%, which is the best result we got among all the methods. It is because neural network is extremely well performance with multiple classes output.

```
Neural Network using raw pixel features:
             precision    recall  f1-score   support

        0.0       0.81      0.78      0.79       659
        1.0       0.58      0.91      0.71       520
        2.0       0.94      0.87      0.90       128
        3.0       0.93      0.73      0.82       309
        4.0       0.90      0.85      0.88       542
        5.0       0.92      0.83      0.87       884
        6.0       0.96      0.80      0.87       138
        7.0       0.89      0.83      0.86       173
        8.0       0.98      0.83      0.90       144
        9.0       0.96      0.89      0.93       208
       10.0       1.00      0.86      0.92        21
       11.0       0.87      0.80      0.83        49

avg / total       0.86      0.83      0.84      3775
```

This result is soooooo awesome!!

Overfitting Avoiding:

For neural network, the biggest problem is overfitting. To avoid overfitting problem, I use python scikit-learn package to compare the accuracy with Spark Multilayer perceptron classifier. The reason I can do this test is scikit-learn package is scikit-learn package has some inner mechanism to avoid overfitting(I used 100 hidden units and 1000 hidden units and same time of iterations—let's say, 50 times—and the accuracy is the same)

Once the accuracy of Spark Multilayer perceptron classifier has the close result to scikit-learn package, I can assume that it has no overfitting in the model.

And Naïve Bayes do not have the problem of overfitting.