CSE 5525

HOMEWORK II – Report

Pragya Arora

arora.170

1. Word's Most Frequent Tag Baseline

Baseline is implemented as follows for part-of-speech tagging

1.1. Train

The model is trained by counting the number of times each word occurs with each tag in the training dataset (twitter train universal.txt)

```
for line in open(sys.argv[1]):
    fields = line.strip().split()
    if fields:
        if dictionary.has_key(fields[0]):
            if dictionary[fields[0]].has_key(fields[1]):
                dictionary[fields[0]][fields[1]] += 1
            else:
                dictionary[fields[0]].update({fields[1]:1})
        else:
            dictionary.setdefault(fields[0],{fields[1]:1})
```

1.2. Test Prediction

For the test dataset (twitter_test_universal.txt) each word in the dataset is tagged by the tag which appears most frequently in the training dataset and the unseen words are tagged as Nouns.

```
for line in open(sys.argv[2]):
    fields = line.strip().split()
    if fields:
        if dictionary.has_key(fields[0]):
            key, value = max(dictionary[fields[0]].iteritems(), key=lambda p: p[1])
            print fields[0],'',key
        else:
            print fields[0],'','NOUN'
    else:
        print ''
```

1.3. Accuracy

The baseline model reported accuracy for twitter_test_universal.txt as **0.800494350282.**

2. Viterbi Algorithm

The Viterbi algorithm for bigram perceptron tagger is implemented as follows. In this algorithm piMatrix is used to calculate the maximum probability of a tag sequence ending in tag at position k and backPointer matrix is used to save the tag sequence.

```
def Viterbi(self, featurizedSentence, theta, slen):
    """Viterbi"""
    piMatrix = np.zeros((theta.shape[0],slen+1))
    backPointer = np.zeros((theta.shape[0],slen+1))
    piMatrix[0][0] = 1
    tagVocabLen = len(self.train.tagVocab.id2word)
    startTag = self.train.tagVocab.GetID('START')

    for c in range(1,tagVocabLen):
        piMatrix[c][1] = featurizedSentence[startTag].dot(theta[startTag][c]) + piMatrix[startTag][startTag]
        backPointer[c][1] = startTag

    for i in range(1,slen):
        for c in range(1,tagVocabLen):
            max = float("-inf")
            for p in range(1,tagVocabLen):
                value = featurizedSentence[i].dot(theta[p][c]) + piMatrix[p][i]
                if max<value:
                    max = value
                    bp = p
            piMatrix[c][i+1] = max
            backPointer[c][i+1] = bp

    max = -1
    bp = -1
    i = slen
    for k in range(1,tagVocabLen):
        if max < piMatrix[k][slen]:
            max = piMatrix[k][slen]
            bp = k

    viterbiSeq = []
    while i > 0:
        viterbiSeq.append(bp)
        bp = backPointer[bp][i]
        i -= 1

    viterbiSeq.reverse()
    return viterbiSeq
```

3. Structured Perceptron

The structured perceptron algorithm is implemented as follows. In this the theta is updated for the tags which are predicted incorrectly.

```
def UpdateTheta(self, sentenceFeatures, goldSequence, viterbiSequence, theta, slen, u, nSent):

    prevTag1 = self.train.tagVocab.GetID('START')
    prevTag2 = prevTag1

    for i in range(slen):
        features = sentenceFeatures[i]
        currTag1 = goldSequence[i]
        currTag2 = viterbiSequence[i]

        if currTag1 != currTag2:
            theta[prevTag1][currTag1] += features
            theta[prevTag2][currTag2] -= features
            u[prevTag1][currTag1] += (features * nSent)
            u[prevTag2][currTag2] -= (features * nSent)

        prevTag1=currTag1
        prevTag2=currTag2
```

For structured perceptron parameter averaging is implemented as follows:

```
def ComputeThetaAverage(self, u, c):
    self.thetaAverage = self.theta - np.divide(u,c)
```

The tags are predicted by using Viterbi algorithm and the theta is updated by using structured perceptron with parameter averaging.

```
def Train(self, nIter):
    u = np.zeros((self.ntags, self.ntags, self.train.vocab.GetVocabSize()))
    count = 1
    for i in range(nIter):
        nSent = 0
        for (s,g) in self.train.featurizedSentences:
            if len(g) <= 1:         #Skip any length 1 sentences - some numerical issues...
                continue
            z = self.Viterbi(s, self.theta, len(g))
            sys.stderr.write("Iteration %s, sentence %s\n" % (i, nSent))
            sys.stderr.write("predicted:\t%s\ngold:\t\t%s\n" % (self.PrintableSequence(z), self.PrintableSequence(g)))
            self.UpdateTheta(s,g,z, self.theta, len(g) ,u,count)
            nSent+=1
            count+=1
    if self.useAveraging:
        self.ComputeThetaAverage(u,count)
```

The accuracy for Twitter for various iterations is as follows:

ITERATIONS	ACCURACY
1	0.811264124294
5	0.853107344633
10	0.854343220339
50	0.852224576271
100	0.850459039548

4. Cross-Domain Experiments

4.1. Train on Penn Treebank and Test on Twitter

The model is trained on ptb_train_universal dataset and tested on twitter_test_universal dataset. The accuracy for various iterations are reported below:

ITERATIONS	ACCURACY (%)
1	0.725988700565
5	0.741525423729
10	0.746468926554
50	0.754237288136
100	0.750882768362

4.2. Train on NPS Chat and Test on Twitter

The model is trained on nps_train_universal dataset and tested on twitter_test_universal dataset. The accuracy for various iterations are reported below:

ITERATIONS	ACCURACY
1	0.788312146893
5	0.819032485876
10	0.813735875706
50	0.813912429379
100	0.811087570621

4.3. Train on Penn Treebank+NPS Chat+Twitter and Test on Twitter

The training data is created by merging twitter_train_universal, nps_train_universal and ptb_train_universal and then the model is trained on the merged data and tested on twitter_test_universal. The accuracy for various iterations are reported below:

ITERATIONS	ACCURACY (%)
1	0.860346045198
5	0.885416666667

10	0.884533898305
50	0.880826271186

4.4. Conclusion

For Cross-Domain experiments we can see that when we train on outside datasets (ptb/nps) the accuracy reduces as the model is tested on the data(twitter) which is unrelated to trained data, so prediction of tags become difficult. But when we train on all the data (twitter+ptb+nps) the accuracy increased from the original model(twitter) as the related data is present in the training dataset and the tags can be predicted.

5. Named Entity Recognition

5.1. Train and Test on NER dataset

The tagger is trained on named entity recognition dataset and the evaluation parameters are reported as follows:

PARAMETER	VALUE
Accuracy	95.73
Precision	48.47
Recall	40.17
F1-measure	43.93

5.2. Additional Features

For the named entity task additional features task features such as lastname, male firstname, female firstname, cities, companies, universities, products, business brands and days are added. The data is scraped form Wikipedia. From the below data we can see that by adding new features the accuracy and various evaluation parameters such as precision, recall and f1-measure have increased.

PARAMETER	VALUE(%)
Accuracy	95.80
Precision	50.00
Recall	46.63
F1-measure	48.26