word2vec 去解决情感分析问题

思路分为两部分，第一步，就是先用 word2vec 和 SGD 训练出每个单词的最优表示向量。第二步，用 Softmax Regression 对训练数据集的每个句子进行训练，得到分类器的参数，用这个参数就可以预测新的数据集的情感分类。其中训练数据集的每个句子，都对应一个 0—1 之间的浮点得分,将这个得分化为 0-4 整数型 5 个级别,分别属于 5 种感情类别，讨厌，有点讨厌，中立，有点喜欢，喜欢。然后将每个句子的词转化成之前训练过的词向量，这样哪些词属于哪个类就知道了，然后用分类器得到分类的边界，得到的参数就可以用来进行预测。

具体实现

第一步，用 word2vec 和 SGD 训练出每个单词的最优表示向量。

执行 c7_run_word2vec.py

其中训练词向量的方法是 c5_word2vec.py

同时用 c6_sgd.py 训练参数，并且将结果保存起来，每 1000 次迭代保存在一个文件中 saved_params_1000.npy

word2vec: 上面提到了，它有两种模型 CBOW 和 Skip-gram，每一种都可以用来训练生成最优的词向量，同时还有两种 cost function 的定义方式，一种是 Softmax cost function， 一种是 Negative sampling cost function，所以在提到 word2vec 的时候，其实是可以有 4 种搭配的方法的，这个小项目里用到的是 Skip-gram 和 Negative sampling cost function 的结合方式。

先定义 skipgram 函数：给一个中心词 currentWord，和它的窗口大小为 2C 的上下文 contextWords，要求出代表它们的词向量矩阵 W1 和 W2。

```
def skipgram(currentWord, C, contextWords, tokens, inputVectors, outputVectors,
    dataset, word2vecCostAndGradient = softmaxCostAndGradient):
    """ Skip-gram model in word2vec """

    currentI = tokens[currentWord]                    #the order of this center word
    predicted = inputVectors[currentI, :]             #turn this word to vector repr

    cost = 0.0
    gradIn = np.zeros(inputVectors.shape)
    gradOut = np.zeros(outputVectors.shape)
    for cwd in contextWords:                          #contextWords is of 2C length
        idx = tokens[cwd]
        cc, gp, gg = word2vecCostAndGradient(predicted, idx, outputVectors, dataset)
        cost += cc                                    #final cost/gradient is the 's
        gradOut += gg
        gradIn[currentI, :] += gp

    return cost, gradIn, gradOut
```

这里用到的成本函数是 Negative sampling，我们的目的就是要使这个成本函数达到最小，然后用这个极值时的参数 grad， 也就是可以得到要求的 wordvectors。要增加准确度，所以可以多次生成中心词和上下文进行训练，然后取平均值，也就是函数 word2vec_sgd_wrapper 做的事情。

```
def negSamplingCostAndGradient(predicted, target, outputVectors, dataset, K=10):
    """ Negative sampling cost function for word2vec models """

    grad = np.zeros(outputVectors.shape)
    gradPred = np.zeros(predicted.shape)

    indices = [target]
    for k in xrange(K):
        newidx = dataset.sampleTokenIdx()
        while newidx == target:
            newidx = dataset.sampleTokenIdx()
        indices += [newidx]

    labels = np.array([1] + [-1 for k in xrange(K)])
    vecs = outputVectors[indices, :]

    t = sigmoid(vecs.dot(predicted) * labels)
    cost = -np.sum(np.log(t))

    delta = labels * (t-1)
    gradPred = delta.reshape((1, K+1)).dot(vecs).flatten()
    gradtemp = delta.reshape((K+1, 1)).dot(predicted.reshape(1, predicted.shape[0]))

    for k in xrange(K+1):
        grad[indices[k]] += gradtemp[k, :]

    return cost, gradPred, grad
```

第二步，用 Softmax Regression 对训练数据集进行分类学习。

执行 c10_sentiment.py

其中用 c6_sgd.py 去训练权重 weights，

然后用 c8_softmaxreg.py 根据训练好的 features,labels,weights 进行类别 label 的预测。

先将数据集分为三部分，training set，deviation set，和 test set。

```
trainset = dataset.getTrainSentences()
devset = dataset.getDevSentences()
testset = dataset.getTestSentences()
```

在 trainset 中，每句话对应一个情感的得分或者说是分类，先将每个 word 在 token 中找到序号，然后在第一步训练好的 wordvectors 中找到相应的词向量。

```
trainFeatures[i, :] = getSentenceFeature(tokens, wordVectors, words)
```

然后用 sgd 和 softmax_wrapper 迭代 10000 次去训练 weights:

weights = sgd(lambda weights: softmax_wrapper(trainFeatures, trainLabels, weights, regularization), weights, 3.0, 10000, PRINT_EVERY=100)

接着用 softmax regression 进行分类的预测:

_, _, pred = softmaxRegression(trainFeatures, trainLabels, weights)

上面用到了不同的 REGULARIZATION＝[0.0, 0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003, 0.01]，在其中选择 accuracy 最好的 REGULARIZATION 和相应的结果：

```
best_dev = 0for result in results:
    if result["dev"] > best_dev:
        best_dev = result["dev"]
        BEST_REGULARIZATION = result["reg"]
        BEST_WEIGHTS = result["weights"]
```

用这个最好的参数在 test set 上进行预测：

```
_, _, pred = softmaxRegression(testFeatures, testLabels, BEST_WEIGHTS)
```

并且的到 accuracy：

```
print "Test accuracy (%%): %f" % accuracy(testLabels, pred)
```

下图是 accuracy 和 REGULARIZATION 在 devset 和 trainset 上的趋势：