#DATASCI W261, Machine Learning at Scale

####Assignement: week #12

Shih Yu Chang

####Due: 2016-12-06, 8AM PST

In [1]:
```
import os
import sys
spark_home = os.environ['SPARK_HOME'] = '/home/cloudera/Downloads/spark-2.
if not spark_home:
    raise ValueError('SPARK_HOME enviroment variable is not set')
sys.path.insert(0,os.path.join(spark_home,'python'))
sys.path.insert(0,os.path.join(spark_home,'python/lib/py4j-0.8.2.1-src.zip
execfile(os.path.join(spark_home,'python/pyspark/shell.py'))
```

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.1
      /_/

Using Python version 2.7.11 (default, Dec  6 2015 18:08:32)
SparkSession available as 'spark'.
```

Part 1

1a

In [2]:
```
# Data for manual OHE
# Note: the first data point does not include any value for the optional t
sampleOne = [(0, 'mouse'), (1, 'black')]
sampleTwo = [(0, 'cat'), (1, 'tabby'), (2, 'mouse')]
sampleThree =  [(0, 'bear'), (1, 'black'), (2, 'salmon')]
sampleDataRDD = sc.parallelize([sampleOne, sampleTwo, sampleThree])
```

In [3]:
```
# TODO: Replace <FILL IN> with appropriate code
sampleOHEDictManual = {}
sampleOHEDictManual[(0,'bear')] = 0 #<FILL IN>
sampleOHEDictManual[(0,'cat')] = 1 #<FILL IN>
sampleOHEDictManual[(0,'mouse')] = 2 #<FILL IN>
sampleOHEDictManual[(1,'black')] = 3 #<FILL IN>
sampleOHEDictManual[(1,'tabby')] = 4 #<FILL IN>
sampleOHEDictManual[(2,'mouse')] = 5 #<FILL IN>
sampleOHEDictManual[(2,'salmon')] = 6 #<FILL IN>
```

In [5]:

```
%%writefile test_helper.py
# A testing helper
#https://pypi.python.org/pypi/test_helper/0.2
import hashlib

class TestFailure(Exception):
  pass
class PrivateTestFailure(Exception):
  pass

class Test(object):
  passed = 0
  numTests = 0
  failFast = False
  private = False

  @classmethod
  def setFailFast(cls):
    cls.failFast = True

  @classmethod
  def setPrivateMode(cls):
    cls.private = True

  @classmethod
  def assertTrue(cls, result, msg=""):
    cls.numTests += 1
    if result == True:
      cls.passed += 1
      print "1 test passed."
    else:
      print "1 test failed. " + msg
      if cls.failFast:
        if cls.private:
          raise PrivateTestFailure(msg)
        else:
          raise TestFailure(msg)

  @classmethod
  def assertEquals(cls, var, val, msg=""):
    cls.assertTrue(var == val, msg)

  @classmethod
  def assertEqualsHashed(cls, var, hashed_val, msg=""):
    cls.assertEquals(cls._hash(var), hashed_val, msg)

  @classmethod
  def printStats(cls):
    print "{0} / {1} test(s) passed.".format(cls.passed, cls.numTests)

  @classmethod
  def _hash(cls, x):
    return hashlib.sha1(str(x)).hexdigest()
```

```
Writing test_helper.py
```

In [6]:

```
# TEST One-hot-encoding (1a)
from test_helper import Test

Test.assertEqualsHashed(sampleOHEDictManual[(0,'bear')],
                        'b6589fc6ab0dc82cf12099d1c2d40ab994e8410c',
                        "incorrect value for sampleOHEDictManual[(0,'bear'
Test.assertEqualsHashed(sampleOHEDictManual[(0,'cat')],
                        '356a192b7913b04c54574d18c28d46e6395428ab',
                        "incorrect value for sampleOHEDictManual[(0,'cat')
Test.assertEqualsHashed(sampleOHEDictManual[(0,'mouse')],
                        'da4b9237bacccdf19c0760cab7aec4a8359010b0',
                        "incorrect value for sampleOHEDictManual[(0,'mouse
Test.assertEqualsHashed(sampleOHEDictManual[(1,'black')],
                        '77de68daecd823babbb58edb1c8e14d7106e83bb',
                        "incorrect value for sampleOHEDictManual[(1,'black
Test.assertEqualsHashed(sampleOHEDictManual[(1,'tabby')],
                        '1b6453892473a467d07372d45eb05abc2031647a',
                        "incorrect value for sampleOHEDictManual[(1,'tabby
Test.assertEqualsHashed(sampleOHEDictManual[(2,'mouse')],
                        'ac3478d69a3c81fa62e60f5c3696165a4e5e6ac4',
                        "incorrect value for sampleOHEDictManual[(2,'mouse
Test.assertEqualsHashed(sampleOHEDictManual[(2,'salmon')],
                        'c1dfd96eea8cc2b62785275bca38ac261256e278',
                        "incorrect value for sampleOHEDictManual[(2,'salmo
Test.assertEquals(len(sampleOHEDictManual.keys()), 7,
                  'incorrect number of keys in sampleOHEDictManual')
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

1b

In [7]:

```
import numpy as np
from pyspark.mllib.linalg import SparseVector
```

In [8]:

```
# TODO: Replace <FILL IN> with appropriate code
aDense = np.array([0., 3., 0., 4.])
aSparse = SparseVector(4,[1,3],[3,4]) #<FILL IN>

bDense = np.array([0., 0., 0., 1.])
bSparse = SparseVector(4,[3],[1]) #<FILL IN>

w = np.array([0.4, 3.1, -1.4, -.5])
print aDense.dot(w)
print aSparse.dot(w)
print bDense.dot(w)
print bSparse.dot(w)
```

```
7.3
7.3
-0.5
-0.5
```

In [9]:

```
# TEST Sparse Vectors (1b)
Test.assertTrue(isinstance(aSparse, SparseVector), 'aSparse needs to be an
Test.assertTrue(isinstance(bSparse, SparseVector), 'aSparse needs to be an
Test.assertTrue(aDense.dot(w) == aSparse.dot(w),
                'dot product of aDense and w should equal dot product of a
Test.assertTrue(bDense.dot(w) == bSparse.dot(w),
                'dot product of bDense and w should equal dot product of b
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

1c

In [10]:

```
# TODO: Replace <FILL IN> with appropriate code
sampleOneOHEFeatManual = SparseVector(7, [2,3] ,[1,1]) #<FILL IN>
sampleTwoOHEFeatManual = SparseVector(7, [1,4,5], [1,1,1]) #<FILL IN>
sampleThreeOHEFeatManual = SparseVector(7, [0,3,6], [1,1,1]) #<FILL IN>
```

In [11]:

```
# TEST OHE Features as sparse vectors (1c)
Test.assertTrue(isinstance(sampleOneOHEFeatManual, SparseVector),
                'sampleOneOHEFeatManual needs to be a SparseVector')
Test.assertTrue(isinstance(sampleTwoOHEFeatManual, SparseVector),
                'sampleTwoOHEFeatManual needs to be a SparseVector')
Test.assertTrue(isinstance(sampleThreeOHEFeatManual, SparseVector),
                'sampleThreeOHEFeatManual needs to be a SparseVector')
Test.assertEqualsHashed(sampleOneOHEFeatManual,
                        'ecc00223d141b7bd0913d52377cee2cf5783abd6',
                        'incorrect value for sampleOneOHEFeatManual')
Test.assertEqualsHashed(sampleTwoOHEFeatManual,
                        '26b023f4109e3b8ab32241938e2e9b9e9d62720a',
                        'incorrect value for sampleTwoOHEFeatManual')
Test.assertEqualsHashed(sampleThreeOHEFeatManual,
                        'c04134fd603ae115395b29dcabe9d0c66fbdc8a7',
                        'incorrect value for sampleThreeOHEFeatManual')
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

1d

In [12]:
```
# TODO: Replace <FILL IN> with appropriate code
def oneHotEncoding(rawFeats, OHEDict, numOHEFeats):
    """Produce a one-hot-encoding from a list of features and an OHE dicti

    Note:
        You should ensure that the indices used to create a SparseVector a

    Args:
        rawFeats (list of (int, str)): The features corresponding to a sin
            feature consists of a tuple of featureID and the feature's val
        OHEDict (dict): A mapping of (featureID, value) to unique integer.
        numOHEFeats (int): The total number of unique OHE features (combin
            value).

    Returns:
        SparseVector: A SparseVector of length numOHEFeats with indicies e
            identifiers for the (featureID, value) combinations that occur
            with values equal to 1.0.
    """
    #<FILL IN>
    # build index array
    ids = [OHEDict[x] for x in rawFeats]
    ones = [1]*len(ids)
    return SparseVector(numOHEFeats, np.sort(ids), ones)

# Calculate the number of features in sampleOHEDictManual
numSampleOHEFeats = len(sampleOHEDictManual) #<FILL IN>

# Run oneHotEnoding on sampleOne
sampleOneOHEFeat = oneHotEncoding(sampleOne, sampleOHEDictManual, numSampl

print sampleOneOHEFeat
```

```
(7,[2,3],[1.0,1.0])
```

In [13]:
```
# TEST Define an OHE Function (1d)
Test.assertTrue(sampleOneOHEFeat == sampleOneOHEFeatManual,
                'sampleOneOHEFeat should equal sampleOneOHEFeatManual')
Test.assertEquals(sampleOneOHEFeat, SparseVector(7, [2,3], [1.0,1.0]),
                  'incorrect value for sampleOneOHEFeat')
Test.assertEquals(oneHotEncoding([(1, 'black'), (0, 'mouse')], sampleOHEDi
                                 numSampleOHEFeats), SparseVector(7, [2,3]
                  'incorrect definition for oneHotEncoding')
```

```
1 test passed.
1 test passed.
1 test passed.
```

1e

In [14]:
```
# TODO: Replace <FILL IN> with appropriate code
sampleOHEData = sampleDataRDD.map(lambda p: oneHotEncoding(p, sampleOHEDic
print sampleOHEData.collect()
```

```
[SparseVector(7, {2: 1.0, 3: 1.0}), SparseVector(7, {1: 1.0, 4: 1.0, 5:
 1.0}), SparseVector(7, {0: 1.0, 3: 1.0, 6: 1.0})]
```

In [15]:
```
# TEST Apply OHE to a dataset (1e)
sampleOHEDataValues = sampleOHEData.collect()
Test.assertTrue(len(sampleOHEDataValues) == 3, 'sampleOHEData should have
Test.assertEquals(sampleOHEDataValues[0], SparseVector(7, {2: 1.0, 3: 1.0}
                  'incorrect OHE for first sample')
Test.assertEquals(sampleOHEDataValues[1], SparseVector(7, {1: 1.0, 4: 1.0,
                  'incorrect OHE for second sample')
Test.assertEquals(sampleOHEDataValues[2], SparseVector(7, {0: 1.0, 3: 1.0,
                  'incorrect OHE for third sample')
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

Part 2

2a

In [16]:
```
# TODO: Replace <FILL IN> with appropriate code
sampleDistinctFeats = sampleDataRDD.flatMap(lambda p: p).distinct().sortBy
# pretty neat it can sort key then value, basically for composite key sort
                         #<FILL IN>)
```

In [17]:
```
# TEST Pair RDD of (featureID, category) (2a)
Test.assertEquals(sorted(sampleDistinctFeats.collect()),
                  [(0, 'bear'), (0, 'cat'), (0, 'mouse'), (1, 'black'),
                   (1, 'tabby'), (2, 'mouse'), (2, 'salmon')],
                  'incorrect value for sampleDistinctFeats')
```

```
1 test passed.
```

2b

In [18]:
```
# TODO: Replace <FILL IN> with appropriate code
sampleOHEDict = sampleDistinctFeats.zipWithIndex().collectAsMap()
print sampleOHEDict
```

```
{(2, 'mouse'): 5, (0, 'cat'): 1, (0, 'bear'): 0, (2, 'salmon'): 6, (1,
'tabby'): 4, (1, 'black'): 3, (0, 'mouse'): 2}
```

In [19]:

```
# TEST OHE Dictionary from distinct features (2b)
Test.assertEquals(sorted(sampleOHEDict.keys()),
                  [(0, 'bear'), (0, 'cat'), (0, 'mouse'), (1, 'black'),
                   (1, 'tabby'), (2, 'mouse'), (2, 'salmon')],
                  'sampleOHEDict has unexpected keys')
Test.assertEquals(sorted(sampleOHEDict.values()), range(7), 'sampleOHEDict
```

```
1 test passed.
1 test passed.
```

2c

In [20]:

```
# TODO: Replace <FILL IN> with appropriate code
def createOneHotDict(inputData):
    """Creates a one-hot-encoder dictionary based on the input data.

    Args:
        inputData (RDD of lists of (int, str)): An RDD of observations whe
            made up of a list of (featureID, value) tuples.

    Returns:
        dict: A dictionary where the keys are (featureID, value) tuples an
            unique integers.
    """
    #<FILL IN>
    return inputData.flatMap(lambda p: p).distinct().sortBy(lambda p: p).z

sampleOHEDictAuto = createOneHotDict(sampleDataRDD) #<FILL IN>
print sampleOHEDictAuto
```

```
{(2, 'mouse'): 5, (0, 'cat'): 1, (0, 'bear'): 0, (2, 'salmon'): 6, (1,
'tabby'): 4, (1, 'black'): 3, (0, 'mouse'): 2}
```

In [21]:

```
# TEST Automated creation of an OHE dictionary (2c)
Test.assertEquals(sorted(sampleOHEDictAuto.keys()),
                  [(0, 'bear'), (0, 'cat'), (0, 'mouse'), (1, 'black'),
                   (1, 'tabby'), (2, 'mouse'), (2, 'salmon')],
                  'sampleOHEDictAuto has unexpected keys')
Test.assertEquals(sorted(sampleOHEDictAuto.values()), range(7),
                  'sampleOHEDictAuto has unexpected values')
```

```
1 test passed.
1 test passed.
```

Part 3

In [22]:

```
# Run this code to view Criteo's agreement
from IPython.lib.display import IFrame

IFrame("http://labs.criteo.com/downloads/2014-kaggle-display-advertising-c
       600, 350)
```

Out[22]:

In [25]:

```
# TODO: Replace <FILL IN> with appropriate code
# Just replace <FILL IN> with the url for dac_sample.tar.gz
import glob
import os.path
import tarfile
import urllib
import urlparse

# Paste url, url should end with: dac_sample.tar.gz
url = 'http://labs.criteo.com/wp-content/uploads/2015/04/dac_sample.tar.gz

url = url.strip()
baseDir = os.path.join('home')
inputPath = os.path.join('cloudera', 'dac_sample.txt')
fileName = os.path.join(baseDir, inputPath)
inputDir = os.path.split(fileName)[0]

def extractTar(check = False):
    # Find the zipped archive and extract the dataset
    tars = glob.glob('dac_sample*.tar.gz*')
    if check and len(tars) == 0:
      return False

    if len(tars) > 0:
        try:
            tarFile = tarfile.open(tars[0])
        except tarfile.ReadError:
            if not check:
                print 'Unable to open tar.gz file.  Check your URL.'
            return False

        tarFile.extract('dac_sample.txt', path=inputDir)
        print 'Successfully extracted: dac_sample.txt'
        return True
    else:
        print 'You need to retry the download with the correct url.'
        print ('Alternatively, you can upload the dac_sample.tar.gz file t
               'directory')
        return False


if os.path.isfile(fileName):
    print 'File is already available. Nothing to do.'
elif extractTar(check = True):
    print 'tar.gz file was already available.'
elif not url.endswith('dac_sample.tar.gz'):
    print 'Check your download url.  Are you downloading the Sample datase
else:
    # Download the file and store it in the same directory as this noteboo
    try:
        urllib.urlretrieve(url, os.path.basename(urlparse.urlsplit(url).pa
    except IOError:
        print 'Unable to download and store: {0}'.format(url)

    extractTar()
```

```
Unable to open tar.gz file.  Check your URL.
```

In [33]:

```
import os.path
#baseDir = os.path.join('home')
#inputPath = os.path.join('cloudera', 'dac_sample.txt')#
#fileName = os.path.join(baseDir, inputPath)
fileName = '/home/cloudera/dac_sample.txt'

if os.path.isfile(fileName):
    rawData = (sc
               .textFile(fileName, 2)
               .map(lambda x: x.replace('\t', ',')))  # work with either '
    print rawData.take(1)
```

```
[u'0,1,1,5,0,1382,4,15,2,181,1,2,,2,68fd1e64,80e26c9b,fb936136,7b4723c4
,25c83c98,7e0ccccf,de7995b8,1f89b562,a73ee510,a8cd5504,b2cb9c98,37c9c16
4,2824a5f6,1adce6ef,8ba8b39a,891b62e7,e5ba7672,f54016b9,21ddcdc9,b1252a
9d,07b5194c,,3a171ecb,c5c50484,e8b83407,9727dd16']
```

3a

In [34]:

```
# TODO: Replace <FILL IN> with appropriate code
weights = [.8, .1, .1]
seed = 42
# Use randomSplit with weights and seed
rawTrainData, rawValidationData, rawTestData = rawData.randomSplit(weights
# Cache the data
#<FILL IN>
rawTrainData.cache()
rawValidationData.cache()
rawTestData.cache()

# count the data
nTrain = rawTrainData.count()
nVal = rawValidationData.count()
nTest = rawTestData.count() #<FILL IN>
print nTrain, nVal, nTest, nTrain + nVal + nTest
print rawData.take(1)
```

```
79911 10075 10014 100000
[u'0,1,1,5,0,1382,4,15,2,181,1,2,,2,68fd1e64,80e26c9b,fb936136,7b4723c4
,25c83c98,7e0ccccf,de7995b8,1f89b562,a73ee510,a8cd5504,b2cb9c98,37c9c16
4,2824a5f6,1adce6ef,8ba8b39a,891b62e7,e5ba7672,f54016b9,21ddcdc9,b1252a
9d,07b5194c,,3a171ecb,c5c50484,e8b83407,9727dd16']
```

In [35]:

```
# TEST Loading and splitting the data (3a)
Test.assertTrue(all([rawTrainData.is_cached, rawValidationData.is_cached,
                'you must cache the split data')
Test.assertEquals(nTrain, 79911, 'incorrect value for nTrain')
Test.assertEquals(nVal, 10075, 'incorrect value for nVal')
Test.assertEquals(nTest, 10014, 'incorrect value for nTest')
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

3b

In [36]:

```
# TODO: Replace <FILL IN> with appropriate code
def parsePoint(point):
    """Converts a comma separated string into a list of (featureID, value)

    Note:
        featureIDs should start at 0 and increase to the number of feature

    Args:
        point (str): A comma separated string where the first value is the
            are features.

    Returns:
        list: A list of (featureID, value) tuples.
    """
    #<FILL IN>
    fea = point.strip().split(',')[1:]
    return [(i, fea[i]) for i in range(len(fea))]

parsedTrainFeat = rawTrainData.map(parsePoint)

numCategories = (parsedTrainFeat
                 .flatMap(lambda x: x)
                 .distinct()
                 .map(lambda x: (x[0], 1))
                 .reduceByKey(lambda x, y: x + y)
                 .sortByKey()
                 .collect())

print numCategories[2][1]

#print parsedTrainFeat.take(2)
```

```
855
```

In [37]:

```
# TEST Extract features (3b)
Test.assertEquals(numCategories[2][1], 855, 'incorrect implementation of p
Test.assertEquals(numCategories[32][1], 4, 'incorrect implementation of pa
```

```
1 test passed.
1 test passed.
```

3c

In [38]:

```
# TODO: Replace <FILL IN> with appropriate code
ctrOHEDict = createOneHotDict(parsedTrainFeat) #<FILL IN>
numCtrOHEFeats = len(ctrOHEDict.keys())
print numCtrOHEFeats
print ctrOHEDict[(0, '')]
```

```
233286
0
```

In [39]:

```
# TEST Create an OHE dictionary from the dataset (3c)
Test.assertEquals(numCtrOHEFeats, 233286, 'incorrect number of features in
Test.assertTrue((0, '') in ctrOHEDict, 'incorrect features in ctrOHEDict')
```

```
1 test passed.
1 test passed.
```

3d

In [40]:

```
from pyspark.mllib.regression import LabeledPoint

# TODO: Replace <FILL IN> with appropriate code
def parseOHEPoint(point, OHEDict, numOHEFeats):
    """Obtain the label and feature vector for this raw observation.

    Note:
        You must use the function `oneHotEncoding` in this implementation
        of this lab may not function as expected.

    Args:
        point (str): A comma separated string where the first value is the
            are features.
        OHEDict (dict of (int, str) to int): Mapping of (featureID, value)
        numOHEFeats (int): The number of unique features in the training d

    Returns:
        LabeledPoint: Contains the label for the observation and the one-h
            raw features based on the provided OHE dictionary.
    """
    #<FILL IN>
    label, fea = point.strip().split(',', 1)
    return LabeledPoint(label, oneHotEncoding(parsePoint(point), OHEDict,

OHETrainData = rawTrainData.map(lambda point: parseOHEPoint(point, ctrOHED
OHETrainData.cache()
print OHETrainData.take(1)

# Check that oneHotEncoding function was used in parseOHEPoint
backupOneHot = oneHotEncoding
oneHotEncoding = None
withOneHot = False
try: parseOHEPoint(rawTrainData.take(1)[0], ctrOHEDict, numCtrOHEFeats)
except TypeError: withOneHot = True
oneHotEncoding = backupOneHot
```

```
[LabeledPoint(0.0, (233286,[2,147,3206,3467,6199,25073,25873,26398,2707
9,28303,28323,28390,28554,28906,29422,65104,75996,87062,87178,93426,944
29,94641,101154,107147,115594,142624,144958,147589,165315,178594,180856
,181114,182200,183160,214088,214101,222510,224951,229867],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0
,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.
0]))]
```

In [41]:
```
# TEST Apply OHE to the dataset (3d)
numNZ = sum(parsedTrainFeat.map(lambda x: len(x)).take(5))
numNZAlt = sum(OHETrainData.map(lambda lp: len(lp.features.indices)).take(
Test.assertEquals(numNZ, numNZAlt, 'incorrect implementation of parseOHEPo
Test.assertTrue(withOneHot, 'oneHotEncoding not present in parseOHEPoint')
```

```
1 test passed.
1 test passed.
```

In [42]:
```
def bucketFeatByCount(featCount):
    """Bucket the counts by powers of two."""
    for i in range(11):
        size = 2 ** i
        if featCount <= size:
            return size
    return -1

featCounts = (OHETrainData
              .flatMap(lambda lp: lp.features.indices)
              .map(lambda x: (x, 1))
              .reduceByKey(lambda x, y: x + y))
featCountsBuckets = (featCounts
                     .map(lambda x: (bucketFeatByCount(x[1]), 1))
                     .filter(lambda (k, v): k != -1)
                     .reduceByKey(lambda x, y: x + y)
                     .collect())
print featCountsBuckets
```

```
[(512, 414), (1024, 255), (2, 24076), (4, 16639), (32, 4755), (64, 2627
), (128, 1476), (256, 748), (16, 7752), (8, 11440), (1, 162813)]
```

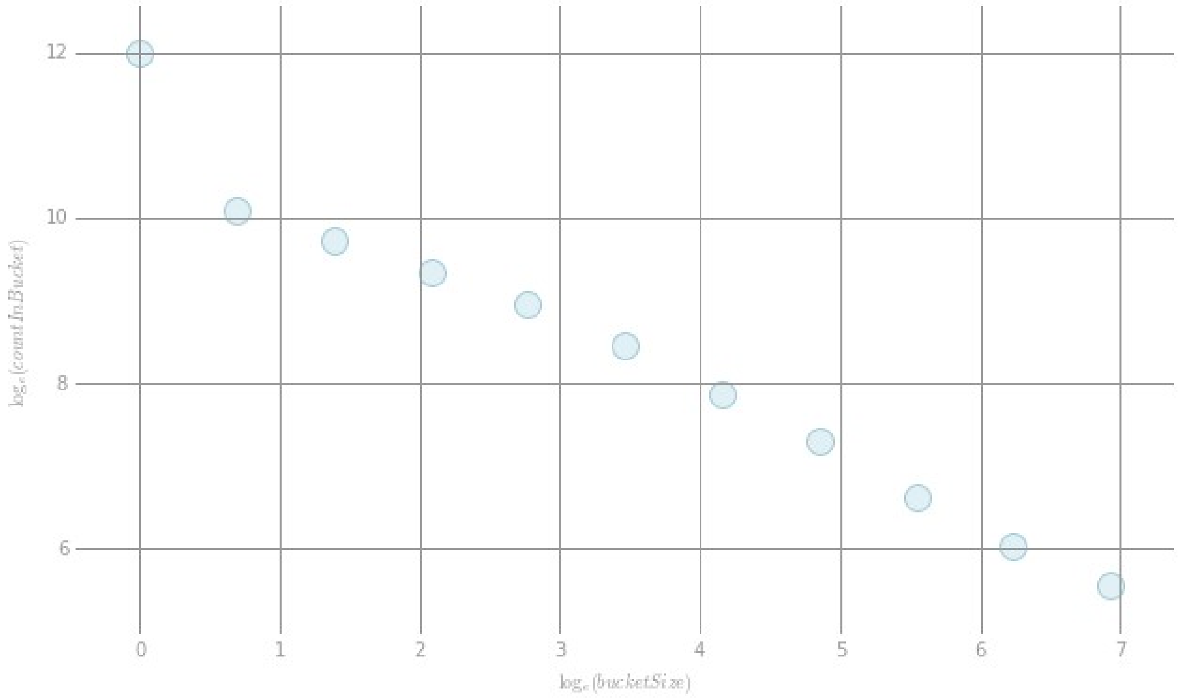
In [43]:

```
import matplotlib.pyplot as plt
%matplotlib inline

x, y = zip(*featCountsBuckets)
x, y = np.log(x), np.log(y)

def preparePlot(xticks, yticks, figsize=(10.5, 6), hideLabels=False, gridC
                gridWidth=1.0):
    """Template for generating the plot layout."""
    plt.close()
    fig, ax = plt.subplots(figsize=figsize, facecolor='white', edgecolor='
    ax.axes.tick_params(labelcolor='#999999', labelsize='10')
    for axis, ticks in [(ax.get_xaxis(), xticks), (ax.get_yaxis(), yticks)
        axis.set_ticks_position('none')
        axis.set_ticks(ticks)
        axis.label.set_color('#999999')
        if hideLabels: axis.set_ticklabels([])
    plt.grid(color=gridColor, linewidth=gridWidth, linestyle='-')
    map(lambda position: ax.spines[position].set_visible(False), ['bottom'
    return fig, ax

# generate layout and plot data
fig, ax = preparePlot(np.arange(0, 10, 1), np.arange(4, 14, 2))
ax.set_xlabel(r'$\log_e(bucketSize)$'), ax.set_ylabel(r'$\log_e(countInBuc
plt.scatter(x, y, s=14**2, c='#d6ebf2', edgecolors='#8cbfd0', alpha=0.75)
#pass
plt.show()
```



3e

In [44]:

```
# TODO: Replace <FILL IN> with appropriate code
def oneHotEncoding(rawFeats, OHEDict, numOHEFeats):
    """Produce a one-hot-encoding from a list of features and an OHE dicti

    Note:
        If a (featureID, value) tuple doesn't have a corresponding key in 
        ignored.

    Args:
        rawFeats (list of (int, str)): The features corresponding to a sin
            feature consists of a tuple of featureID and the feature's val
        OHEDict (dict): A mapping of (featureID, value) to unique integer.
        numOHEFeats (int): The total number of unique OHE features (combin
            value).

    Returns:
        SparseVector: A SparseVector of length numOHEFeats with indicies e
            identifiers for the (featureID, value) combinations that occur
            with values equal to 1.0.
    """
    #<FILL IN>
    ids = [OHEDict[x] for x in rawFeats if x in OHEDict]
    ones = [1]*len(ids)
    return SparseVector(numOHEFeats, np.sort(ids), ones)

OHEValidationData = rawValidationData.map(lambda point: parseOHEPoint(poin
OHEValidationData.cache()
print OHEValidationData.take(1)
```

```
[LabeledPoint(0.0, (233286,[0,1970,3117,3577,9095,24758,26026,26425,268
63,28301,28311,28390,28681,28709,29599,58983,82008,87062,87178,93196,94
413,94641,96215,104657,137971,142919,144970,149229,164814,178594,179212
,180974,182197,182216,214088,214100,216589,224912,224961],[1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0
,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.
0]))]
```

In [45]:

```
# TEST Handling unseen features (3e)
numNZVal = (OHEValidationData
            .map(lambda lp: len(lp.features.indices))
            .sum())
Test.assertEquals(numNZVal, 372080, 'incorrect number of features')
```

```
1 test passed.
```

Part 4

4a

In [46]:

```
from pyspark.mllib.classification import LogisticRegressionWithSGD

# fixed hyperparameters
numIters = 50
stepSize = 10.
regParam = 1e-6
regType = 'l2'
includeIntercept = True
```

In [47]:

```
# TODO: Replace <FILL IN> with appropriate code
model0 = LogisticRegressionWithSGD.train(OHETrainData,
                                         iterations=numIters,
                                         step=stepSize,
                                         regParam=regParam,
                                         regType=regType,
                                         intercept=includeIntercept) #<FIL
sortedWeights = sorted(model0.weights)
print sortedWeights[:5], model0.intercept
```

```
/home/cloudera/Downloads/spark-2.0.1-bin-hadoop2.6/python/pyspark/mllib
/classification.py:313: UserWarning: Deprecated in 2.0.0. Use ml.classi
fication.LogisticRegression or LogisticRegressionWithLBFGS.
  "Deprecated in 2.0.0. Use ml.classification.LogisticRegression or "

[-0.45899236853575626, -0.37973707648623972, -0.36996558266753299, -0.3
6934962879928268, -0.32697945415010637] 0.56455084025
```

In [48]:

```
# TEST Logistic regression (4a)
Test.assertTrue(np.allclose(model0.intercept,  0.56455084025), 'incorrect 
Test.assertTrue(np.allclose(sortedWeights[0:5],
                [-0.45899236853575609, -0.37973707648623956, -0.3699655826
                 -0.36934962879928263, -0.32697945415010637]), 'incorrect 
```

```
1 test passed.
1 test passed.
```

4b

In [49]:

```
# TODO: Replace <FILL IN> with appropriate code
from math import log

def computeLogLoss(p, y):
    """Calculates the value of log loss for a given probabilty and label.

    Note:
        log(0) is undefined, so when p is 0 we need to add a small value (
        and when p is 1 we need to subtract a small value (epsilon) from i

    Args:
        p (float): A probabilty between 0 and 1.
        y (int): A label.  Takes on the values 0 and 1.

    Returns:
        float: The log loss value.
    """
    epsilon = 10e-12
    #<FILL IN>
    return -log(p+epsilon) if y==1 else -log(1-p+epsilon)

print computeLogLoss(.5, 1)
print computeLogLoss(.5, 0)
print computeLogLoss(.99, 1)
print computeLogLoss(.99, 0)
print computeLogLoss(.01, 1)
print computeLogLoss(.01, 0)
print computeLogLoss(0, 1)
print computeLogLoss(1, 1)
print computeLogLoss(1, 0)
```

```
0.69314718054
0.69314718054
0.0100503358434
4.60517018499
4.60517018499
0.0100503358434
25.3284360229
-1.00000008274e-11
25.3284360229
```

In [50]:

```
# TEST Log loss (4b)
Test.assertTrue(np.allclose([computeLogLoss(.5, 1), computeLogLoss(.01, 0)
                            [0.69314718056, 0.0100503358535, 4.60517018599
                'computeLogLoss is not correct')
Test.assertTrue(np.allclose([computeLogLoss(0, 1), computeLogLoss(1, 1), c
                            [25.3284360229, 1.00000008275e-11, 25.32843602
                'computeLogLoss needs to bound p away from 0 and 1 by epsi
```

```
1 test passed.
1 test passed.
```

4c

In [51]:

```
# TODO: Replace <FILL IN> with appropriate code
# Note that our dataset has a very high click-through rate by design
# In practice click-through rate can be one to two orders of magnitude low
classOneFracTrain = OHETrainData.map(lambda p: p.label).mean() #<FILL IN>
print classOneFracTrain

logLossTrBase = OHETrainData.map(lambda p: computeLogLoss(classOneFracTrai
print 'Baseline Train Logloss = {0:.3f}\n'.format(logLossTrBase)
```

```
0.22717773523
Baseline Train Logloss = 0.536
```

In [52]:

```
# TEST Baseline log loss (4c)
Test.assertTrue(np.allclose(classOneFracTrain, 0.22717773523), 'incorrect 
Test.assertTrue(np.allclose(logLossTrBase, 0.535844), 'incorrect value for
```

```
1 test passed.
1 test passed.
```

4d

In [53]:

```
# TODO: Replace <FILL IN> with appropriate code
from math import exp #  exp(-t) = e^-t

def getP(x, w, intercept):
    """Calculate the probability for an observation given a set of weights

    Note:
        We'll bound our raw prediction between 20 and -20 for numerical pu

    Args:
        x (SparseVector): A vector with values of 1.0 for features that ex
            observation and 0.0 otherwise.
        w (DenseVector): A vector of weights (betas) for the model.
        intercept (float): The model's intercept.

    Returns:
        float: A probability between 0 and 1.
    """
    rawPrediction = x.dot(w) + intercept #<FILL IN>

    # Bound the raw prediction value
    rawPrediction = min(rawPrediction, 20)
    rawPrediction = max(rawPrediction, -20)
    return 1/(1+exp(-rawPrediction)) #<FILL IN>

trainingPredictions = OHETrainData.map(lambda p: getP(p.features, model0.w

print trainingPredictions.take(5)
```

```
[0.30262882023911114, 0.10362661997434088, 0.2836342478387561, 0.178461
0205788012, 0.5389775379218853]
```

In [54]:

```
# TEST Predicted probability (4d)
Test.assertTrue(np.allclose(trainingPredictions.sum(), 18135.4834348),
                 'incorrect value for trainingPredictions')
```

```
1 test passed.
```

4e

In [55]:

```
# TODO: Replace <FILL IN> with appropriate code
def evaluateResults(model, data):
    """Calculates the log loss for the data given the model.

    Args:
        model (LogisticRegressionModel): A trained logistic regression mod
        data (RDD of LabeledPoint): Labels and features for each observati

    Returns:
        float: Log loss for the data.
    """
    #<FILL IN>
    return data.map(lambda p: computeLogLoss(getP(p.features, model.weight


logLossTrLR0 = evaluateResults(model0, OHETrainData)
print ('OHE Features Train Logloss:\n\tBaseline = {0:.3f}\n\tLogReg = {1:.
       .format(logLossTrBase, logLossTrLR0))
```

```
OHE Features Train Logloss:
        Baseline = 0.536
        LogReg = 0.456903
```

In [56]:

```
# TEST Evaluate the model (4e)
Test.assertTrue(np.allclose(logLossTrLR0, 0.456903), 'incorrect value for
```

```
1 test passed.
```

4f

In [57]:

```
# TODO: Replace <FILL IN> with appropriate code
logLossValBase = OHEValidationData.map(lambda p: computeLogLoss(classOneFr

logLossValLR0 = evaluateResults(model0, OHEValidationData) #<FILL IN>
print ('OHE Features Validation Logloss:\n\tBaseline = {0:.3f}\n\tLogReg =
       .format(logLossValBase, logLossValLR0))
```

```
OHE Features Validation Logloss:
        Baseline = 0.528
        LogReg = 0.457
```

In [58]:

```
# TEST Validation log loss (4f)
Test.assertTrue(np.allclose(logLossValBase, 0.527603), 'incorrect value fo
Test.assertTrue(np.allclose(logLossValLR0, 0.456957), 'incorrect value for
```

```
1 test passed.
1 test passed.
```

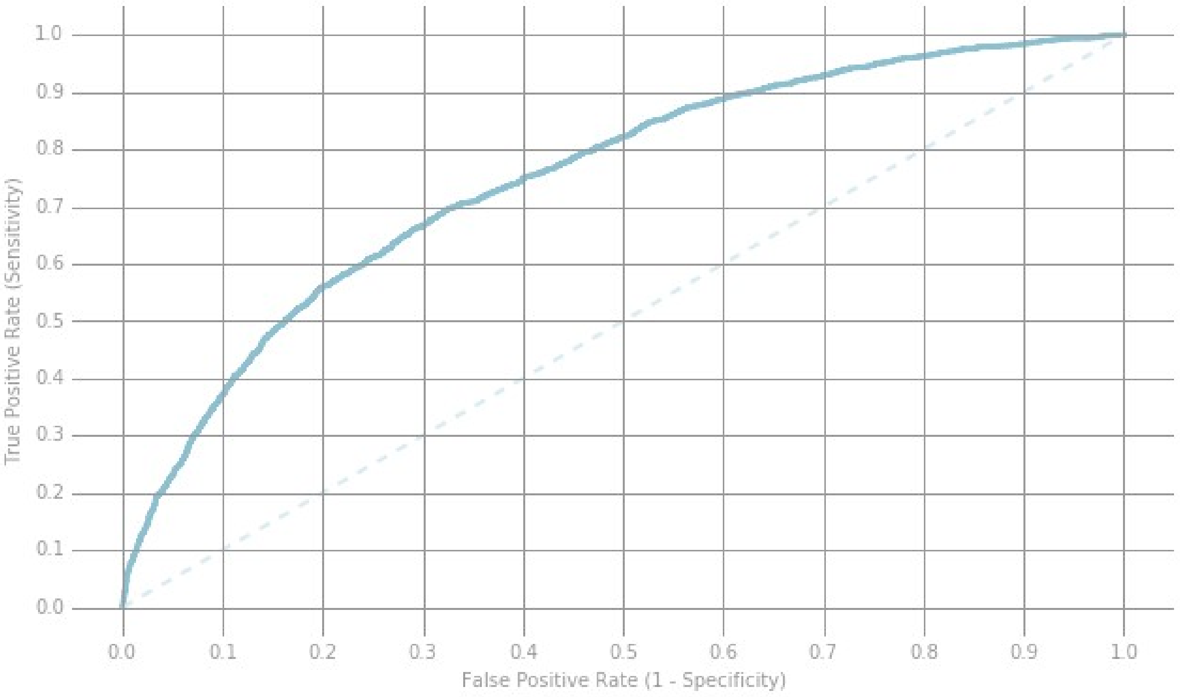
In [59]:

```
labelsAndScores = OHEValidationData.map(lambda lp:
                                            (lp.label, getP(lp.features, m
labelsAndWeights = labelsAndScores.collect()
labelsAndWeights.sort(key=lambda (k, v): v, reverse=True)
labelsByWeight = np.array([k for (k, v) in labelsAndWeights])

length = labelsByWeight.size
truePositives = labelsByWeight.cumsum()
numPositive = truePositives[-1]
falsePositives = np.arange(1.0, length + 1, 1.) - truePositives

truePositiveRate = truePositives / numPositive
falsePositiveRate = falsePositives / (length - numPositive)

# Generate layout and plot data
fig, ax = preparePlot(np.arange(0., 1.1, 0.1), np.arange(0., 1.1, 0.1))
ax.set_xlim(-.05, 1.05), ax.set_ylim(-.05, 1.05)
ax.set_ylabel('True Positive Rate (Sensitivity)')
ax.set_xlabel('False Positive Rate (1 - Specificity)')
plt.plot(falsePositiveRate, truePositiveRate, color='#8cbfd0', linestyle='
plt.plot((0., 1.), (0., 1.), linestyle='--', color='#d6ebf2', linewidth=2.
pass
```



Part 5

5a

In [61]:

```
from collections import defaultdict
import hashlib

def hashFunction(numBuckets, rawFeats, printMapping=False):
    """Calculate a feature dictionary for an observation's features based

    Note:
        Use printMapping=True for debug purposes and to better understand

    Args:
        numBuckets (int): Number of buckets to use as features.
        rawFeats (list of (int, str)): A list of features for an observati
            (featureID, value) tuples.
        printMapping (bool, optional): If true, the mappings of featureStr
            printed.

    Returns:
        dict of int to float:  The keys will be integers which represent t
            features have been hashed to.  The value for a given key will
            (featureID, value) tuples that have hashed to that key.
    """
    mapping = {}
    for ind, category in rawFeats:
        featureString = category + str(ind)
        mapping[featureString] = int(int(hashlib.md5(featureString).hexdig
    if(printMapping): print mapping
    sparseFeatures = defaultdict(float)
    for bucket in mapping.values():
        sparseFeatures[bucket] += 1.0
    return dict(sparseFeatures)
```

In [62]:

```
# TODO: Replace <FILL IN> with appropriate code
# Use four buckets
sampOneFourBuckets = hashFunction(4, sampleOne, True)
sampTwoFourBuckets = hashFunction(4, sampleTwo, True)
sampThreeFourBuckets = hashFunction(4, sampleThree, True)

# Use one hundred buckets
sampOneHundredBuckets = hashFunction(100, sampleOne, True)
sampTwoHundredBuckets = hashFunction(100, sampleTwo, True)
sampThreeHundredBuckets = hashFunction(100, sampleThree, True)

print '\t\t 4 Buckets \t\t\t 100 Buckets'
print 'SampleOne:\t {0}\t\t {1}'.format(sampOneFourBuckets, sampOneHundred
print 'SampleTwo:\t {0}\t\t {1}'.format(sampTwoFourBuckets, sampTwoHundred
print 'SampleThree:\t {0}\t {1}'.format(sampThreeFourBuckets, sampThreeHun
```

```
{'black1': 2, 'mouse0': 3}
{'cat0': 0, 'tabby1': 0, 'mouse2': 2}
{'bear0': 0, 'black1': 2, 'salmon2': 1}
{'black1': 14, 'mouse0': 31}
{'cat0': 40, 'tabby1': 16, 'mouse2': 62}
{'bear0': 72, 'black1': 14, 'salmon2': 5}
                 4 Buckets                        100 Buckets
SampleOne:       {2: 1.0, 3: 1.0}                 {14: 1.0, 31: 1.0}
SampleTwo:       {0: 2.0, 2: 1.0}                 {40: 1.0, 16: 1.0, 62:
 1.0}
SampleThree:     {0: 1.0, 1: 1.0, 2: 1.0}         {72: 1.0, 5: 1.0, 14:
1.0}
```

In [63]:

```
# TEST Hash function (5a)
Test.assertEquals(sampOneFourBuckets, {2: 1.0, 3: 1.0}, 'incorrect value f
Test.assertEquals(sampThreeHundredBuckets, {72: 1.0, 5: 1.0, 14: 1.0},
                  'incorrect value for sampThreeHundredBuckets')
```

```
1 test passed.
1 test passed.
```

5b

In [64]:

```
# TODO: Replace <FILL IN> with appropriate code
def parseHashPoint(point, numBuckets):
    """Create a LabeledPoint for this observation using hashing.

    Args:
        point (str): A comma separated string where the first value is the
            features.
        numBuckets: The number of buckets to hash to.

    Returns:
        LabeledPoint: A LabeledPoint with a label (0.0 or 1.0) and a Spars
            features.
    """
    # <FILL IN>
    elem = point.strip().split(',')
    rawFea = [(i, elem[i+1]) for i in range(len(elem) - 1)]
    index = np.sort(hashFunction(numBuckets, rawFea, False).keys())
    return LabeledPoint(elem[0], SparseVector(numBuckets, index, [1]*len(i


numBucketsCTR = 2 ** 15
hashTrainData = rawTrainData.map(lambda p: parseHashPoint(p, numBucketsCTR
hashTrainData.cache()
hashValidationData = rawValidationData.map(lambda p: parseHashPoint(p, num
hashValidationData.cache()
hashTestData = rawTestData.map(lambda p: parseHashPoint(p, numBucketsCTR))
hashTestData.cache()

print hashTrainData.take(1)
```

```
[LabeledPoint(0.0, (32768,[1305,2883,3807,4814,4866,4913,6952,7117,9985
,10316,11512,11722,12365,13893,14735,15816,16198,17761,19274,21604,2225
6,22563,22785,24855,25202,25533,25721,26487,26656,27668,28211,29152,294
02,29873,30039,31484,32493,32708],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0
,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]))]
```

In [65]:

```
# TEST Creating hashed features (5b)
hashTrainDataFeatureSum = sum(hashTrainData
                              .map(lambda lp: len(lp.features.indices))
                              .take(20))
hashTrainDataLabelSum = sum(hashTrainData
                            .map(lambda lp: lp.label)
                            .take(100))
hashValidationDataFeatureSum = sum(hashValidationData
                                   .map(lambda lp: len(lp.features.indices))
                                   .take(20))
hashValidationDataLabelSum = sum(hashValidationData
                                 .map(lambda lp: lp.label)
                                 .take(100))
hashTestDataFeatureSum = sum(hashTestData
                             .map(lambda lp: len(lp.features.indices))
                             .take(20))
hashTestDataLabelSum = sum(hashTestData
                           .map(lambda lp: lp.label)
                           .take(100))

Test.assertEquals(hashTrainDataFeatureSum, 772, 'incorrect number of featu
Test.assertEquals(hashTrainDataLabelSum, 24.0, 'incorrect labels in hashTr
Test.assertEquals(hashValidationDataFeatureSum, 776,
                  'incorrect number of features in hashValidationData')
Test.assertEquals(hashValidationDataLabelSum, 16.0, 'incorrect labels in h
Test.assertEquals(hashTestDataFeatureSum, 774, 'incorrect number of featur
Test.assertEquals(hashTestDataLabelSum, 23.0, 'incorrect labels in hashTes
```

```
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
1 test passed.
```

5c

In [66]:

```
# TODO: Replace <FILL IN> with appropriate code
def computeSparsity(data, d, n):
    """Calculates the average sparsity for the features in an RDD of Label

    Args:
        data (RDD of LabeledPoint): The LabeledPoints to use in the sparsi
        d (int): The total number of features.
        n (int): The number of observations in the RDD.

    Returns:
        float: The average of the ratio of features in a point to total fe
    """
    #<FILL IN>
    return data.map(lambda p: 1.0*len(p.features.indices)/d).mean()

averageSparsityHash = computeSparsity(hashTrainData, numBucketsCTR, nTrain
averageSparsityOHE = computeSparsity(OHETrainData, numCtrOHEFeats, nTrain)

print 'Average OHE Sparsity: {0:.7e}'.format(averageSparsityOHE)
print 'Average Hash Sparsity: {0:.7e}'.format(averageSparsityHash)
```

```
Average OHE Sparsity: 1.6717677e-04
Average Hash Sparsity: 1.1805561e-03
```

In [67]:

```
# TEST Sparsity (5c)
Test.assertTrue(np.allclose(averageSparsityOHE, 1.6717677e-04),
                'incorrect value for averageSparsityOHE')
Test.assertTrue(np.allclose(averageSparsityHash, 1.1805561e-03),
                'incorrect value for averageSparsityHash')
```

```
1 test passed.
1 test passed.
```

5d

In [68]:

```
numIters = 500
regType = 'l2'
includeIntercept = True

# Initialize variables using values from initial model training
bestModel = None
bestLogLoss = 1e10
```

In [69]:

```
# TODO: Replace <FILL IN> with appropriate code
stepSizes = [1, 10] #<FILL IN>
regParams = [1e-6, 1e-3] #<FILL IN>
for stepSize in stepSizes:
    for regParam in regParams:
        model = (LogisticRegressionWithSGD
                 .train(hashTrainData, numIters, stepSize, regParam=regPar
                        intercept=includeIntercept))
        logLossVa = evaluateResults(model, hashValidationData)
        print ('\tstepSize = {0:.1f}, regParam = {1:.0e}: logloss = {2:.6f
               .format(stepSize, regParam, logLossVa))
        if (logLossVa < bestLogLoss):
            bestModel = model
            bestLogLoss = logLossVa

print ('Hashed Features Validation Logloss:\n\tBaseline = {0:.6f}\n\tLogRe
       .format(logLossValBase, bestLogLoss))
```

```
        stepSize = 1.0, regParam = 1e-06: logloss = 0.474694
        stepSize = 1.0, regParam = 1e-03: logloss = 0.474999
        stepSize = 10.0, regParam = 1e-06: logloss = 0.449679
        stepSize = 10.0, regParam = 1e-03: logloss = 0.451841
Hashed Features Validation Logloss:
        Baseline = 0.527603
        LogReg = 0.449679
```

In [70]:

```
# TEST Logistic model with hashed features (5d)
Test.assertTrue(np.allclose(bestLogLoss, 0.4481683608), 'incorrect value f
```

```
1 test failed. incorrect value for bestLogLoss
```

Visualization

In [71]:

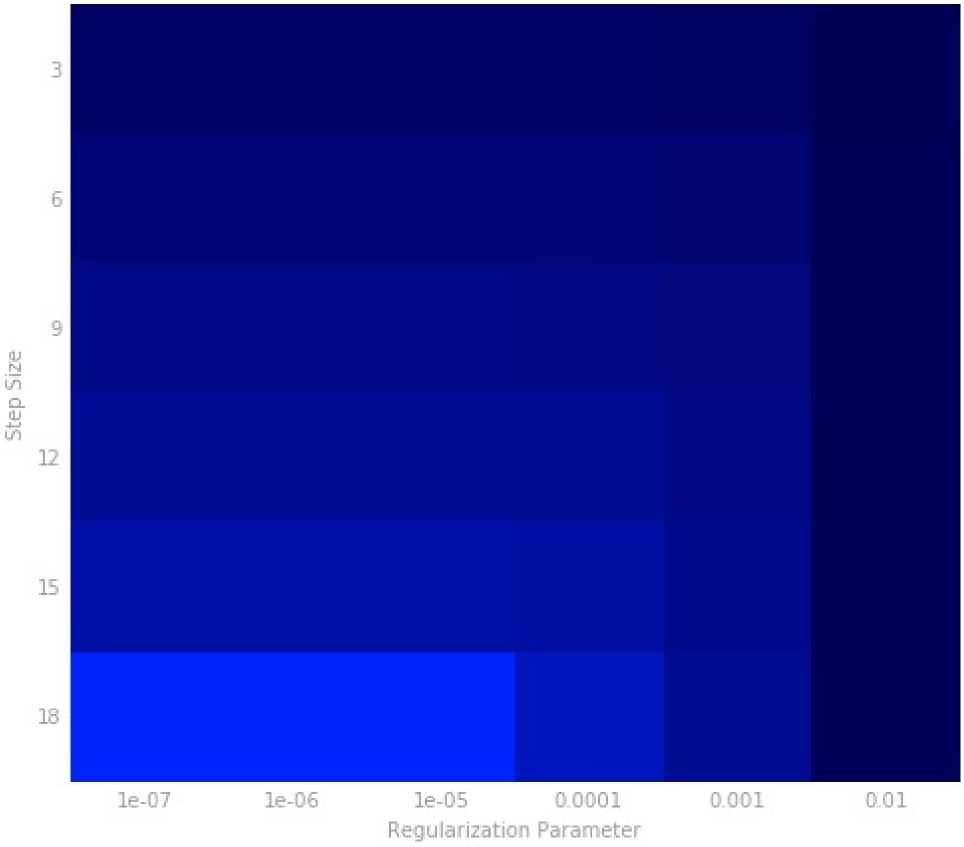
```
from matplotlib.colors import LinearSegmentedColormap

# Saved parameters and results.  Eliminate the time required to run 36 mod
stepSizes = [3, 6, 9, 12, 15, 18]
regParams = [1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2]
logLoss = np.array([[ 0.45808431,  0.45808493,  0.45809113,  0.45815333,
                    [ 0.45188196,  0.45188306,  0.4518941,   0.4520051,
                    [ 0.44886478,  0.44886613,  0.44887974,  0.44902096,
                    [ 0.44706645,  0.4470698,   0.44708102,  0.44724251,
                    [ 0.44588848,  0.44589365,  0.44590568,  0.44606631,
                    [ 0.44508948,  0.44509474,  0.44510274,  0.44525007,

numRows, numCols = len(stepSizes), len(regParams)
logLoss = np.array(logLoss)
logLoss.shape = (numRows, numCols)

fig, ax = preparePlot(np.arange(0, numCols, 1), np.arange(0, numRows, 1),
                      hideLabels=True, gridWidth=0.)
ax.set_xticklabels(regParams), ax.set_yticklabels(stepSizes)
ax.set_xlabel('Regularization Parameter'), ax.set_ylabel('Step Size')

colors = LinearSegmentedColormap.from_list('blue', ['#0022ff', '#000055'],
image = plt.imshow(logLoss,interpolation='nearest', aspect='auto',
                    cmap = colors)
#pass
plt.show()
```



5e

In [72]:

```
# TODO: Replace <FILL IN> with appropriate code
# Log loss for the best model from (5d)
best = (LogisticRegressionWithSGD.train(hashTrainData,
                                        iterations = numIters,
                                        step = 17,
                                        regParam=1e-05,
                                        regType=regType,
                                        intercept=includeIntercept))

logLossTest = evaluateResults(best, hashTestData) #<FILL IN>

# Log loss for the baseline model
logLossTestBaseline = hashTestData.map(lambda p: computeLogLoss(classOneFr

print ('Hashed Features Test Log Loss:\n\tBaseline = {0:.6f}\n\tLogReg = {
       .format(logLossTestBaseline, logLossTest))
```

```
Hashed Features Test Log Loss:
        Baseline = 0.537438
        LogReg = 0.453568
```

In [73]:

```
# TEST Evaluate on the test set (5e)
Test.assertTrue(np.allclose(logLossTestBaseline, 0.537438),
                'incorrect value for logLossTestBaseline')
Test.assertTrue(np.allclose(logLossTest, 0.455616931), 'incorrect value fo
```

```
1 test passed.
1 test failed. incorrect value for logLossTest
```

In [ ]: