In [13]:

```
##### Cooktime Prediction
dataDir = "/Users/Judy-Ccino412/Desktop/cookdata"

def readGz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)

def readCSV(path):
    f = gzip.open(path, 'rt')
    c = csv.reader(f)
    header = next(c)
    for l in c:
        d = dict(zip(header,l))
        yield d['user_id'],d['recipe_id'],d

data = []
for d in readGz(dataDir + 'trainRecipes.json.gz'):
    data.append(d)

# few utility features
allRatings = []
userRatings = defaultdict(list)
data = []
for user,recipe,d in readCSV(dataDir + "trainInteractions.csv.gz"):
    data.append(d)
    r = int(d['rating'])
    allRatings.append(r)
    userRatings[user].append(r)

r_data = {}
mins_data = {}
steps_data = {}
for d in readGz("trainRecipes.json.gz"):
    r = d['recipe_id']
    i = d['ingredients']
    s = d['steps']
    mi = d['minutes']
    r_data[r] = i
    mins_data[r] = mi
    steps_data[r] = s

# Reviews 1-190,000 for training
training = data[:190000]

# Ignore capitalization and remove punctuation
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in training:
    r = ''.join([c for c in d['steps'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1
```

```
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

# 4,000 most common words in the training set
bigger_words = [x[1] for x in counts[:4000]]
wordId = dict(zip(bigger_words, range(len(bigger_words))))
wordSet = set(bigger_words)

# Build bag-of-words feature vectors by counting the instances of these 4,000
def feature(datum):
    feat = [0]*len(bigger_words)
    r = ''.join([c for c in datum['steps'].lower() if not c in punctuation])
    for w in r.split():
        if w in bigger_words:
            feat[wordId[w]] += 1
    feat.append(1) # offset
    return feat

# Extract bag-of-word features in training
X_train = [feature(d) for d in training]
y_train = [d['minutes'] for d in training]

pl = Pipeline([('regressor', linear_model.Ridge(alpha = 1.0,
                                                fit_intercept=False,
                                                normalize = False))])
parameters = {'regressor__alpha': [200, 230, 250, 280, 320, 400]}

# I used grid search to find the best alpha = 400 for Ridge regression
grids = GridSearchCV(pl, param_grid=parameters, cv=4, return_train_score=True
grids.fit(X_train, y_train);
grids.best_params_['regressor__alpha'] # alpha = 400
```

In [43]:

```
#### fit regressor
clf = linear_model.Ridge(400, fit_intercept=False) # Regularized regression:
clf.fit(X_train, y_train)

#### predict on test set
predictions = open("predictions_Minutes.txt", 'w')
predictions.write("recipe_id,prediction\n")
for d in readGz("testRecipes.json.gz"):
    x = feature(d)
    pred = clf.predict([x])[0]
    # if there is a negative prediction, predict 30 min instead (which is clo
    if pred < 0:
        pred = 30
        predictions.write(d['recipe_id'] + ',' + str(pred) + '\n')

predictions.close()
```

Out[43]:

```
Ridge(alpha=400, fit_intercept=False)
```