David Foor
CS 156A Kaggle Competition
Due 2 26, 2014

1 Overview

Below I outline the various tests I ran in the competition, each test has it's own approach and thus its own overview, data, learning, and model selection components.

1 No Learning

1 Overview

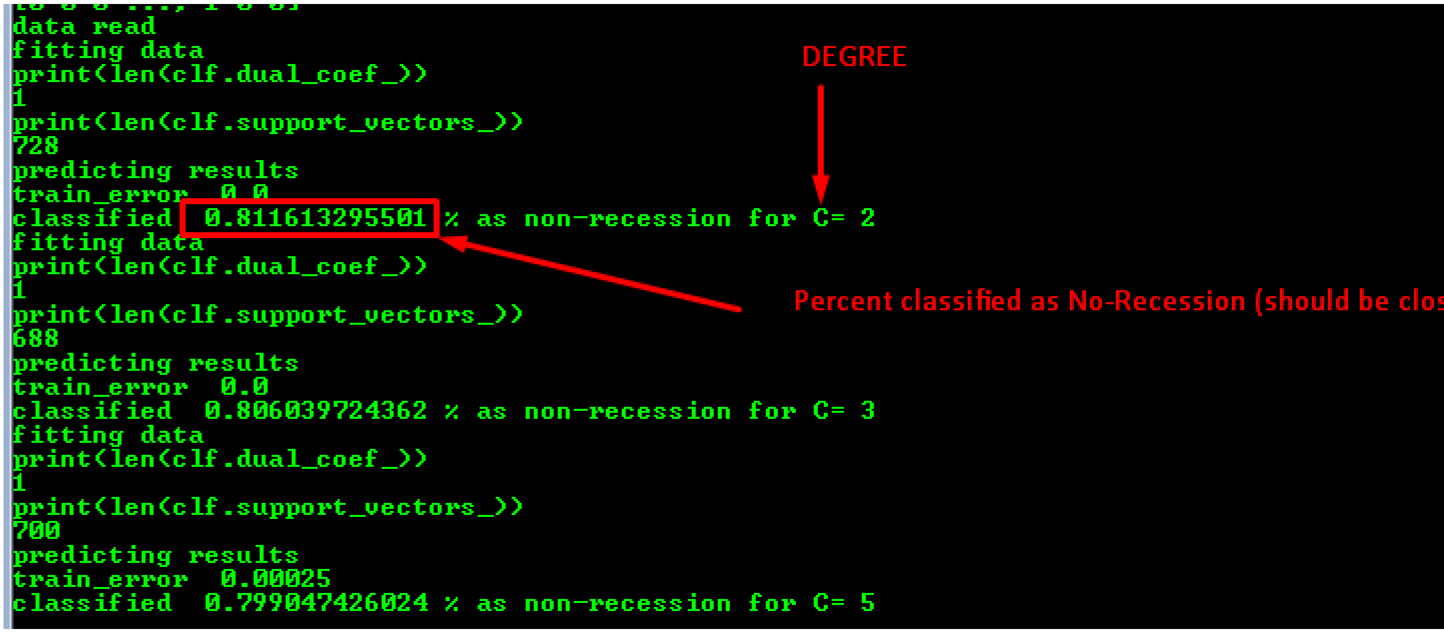
The first submissions I made included a random output selection, which returned 49% correctness (surprise surprise), and an all zeros submission, which garnered an 83% accuracy. This test gave me information about the distribution of recession to no-recession reports. Thus off the bat, and assuming that the hidden test set is of the same distribution (we were told that it is), I should expect for a good data train, that the learning algorithm classifies 83% of the data into the no-recession catagory.

2 Data Manipulation

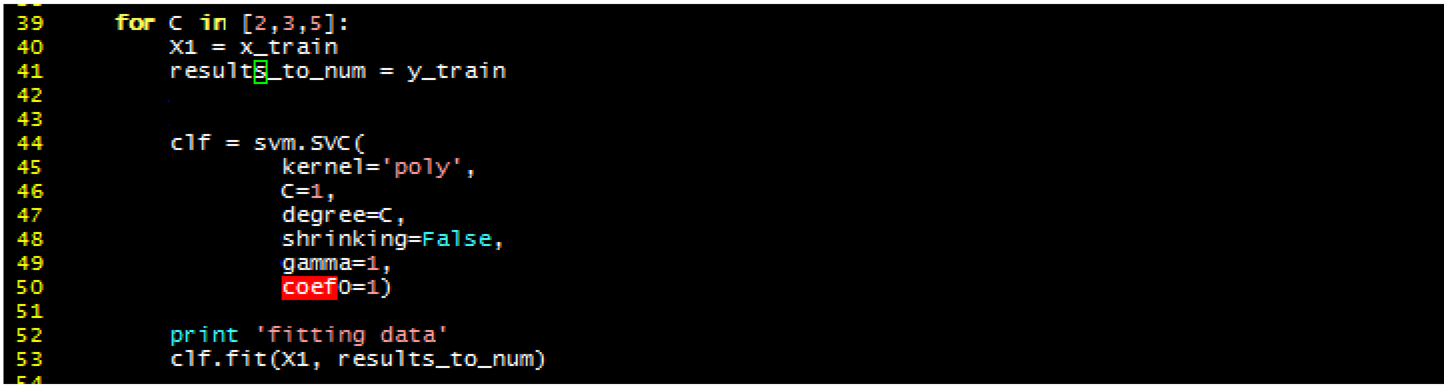
3 Learning Algorithm

4 Model Selection

SVM implementation using polynomials of varying degrees:

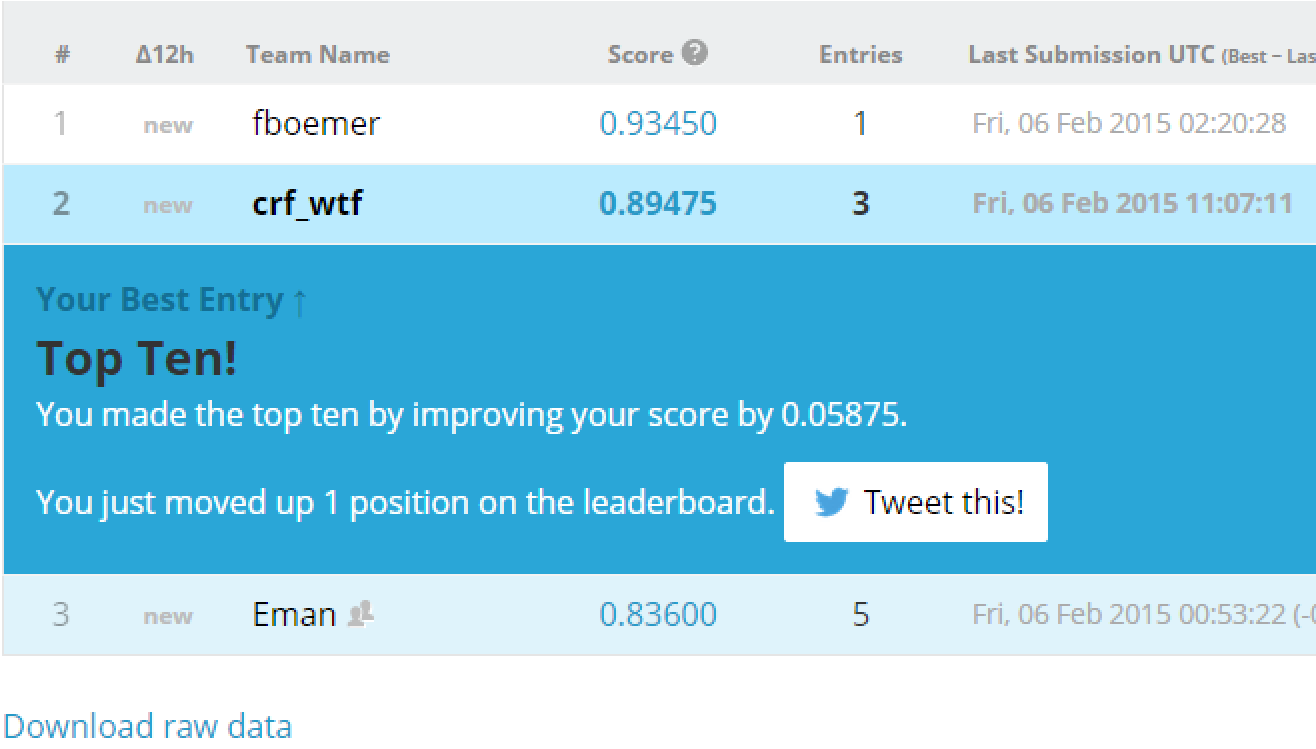


```
[0 0 0 ..., 1 0 0]
data read
fitting data
print(len(clf.dual_coef_))
1
print(len(clf.support_vectors_))
728
predicting results
train_error  0.0
classified  0.811613295501 % as non-recession for C= 2
fitting data
print(len(clf.dual_coef_))
1
print(len(clf.support_vectors_))
688
predicting results
train_error  0.0
classified  0.806039724362 % as non-recession for C= 3
fitting data
print(len(clf.dual_coef_))
1
print(len(clf.support_vectors_))
700
predicting results
train_error  0.00025
classified  0.799047426024 % as non-recession for C= 5
```



```
    for C in [2,3,5]:
        X1 = x_train
        results_to_num = y_train


        clf = svm.SVC(
                kernel='poly',
                C=1,
                degree=C,
                shrinking=False,
                gamma=1,
                coef0=1)

        print 'fitting data'
        clf.fit(X1, results_to_num)
```

Submitted deg 2, received 89%:



#	Δ12h	Team Name	Score	Entries	Last Submission UTC (Best – Las
1	new	fboemer	0.93450	1	Fri, 06 Feb 2015 02:20:28
2	new	**crf_wtf**	**0.89475**	**3**	**Fri, 06 Feb 2015 11:07:11**

Your Best Entry ↑

Top Ten!

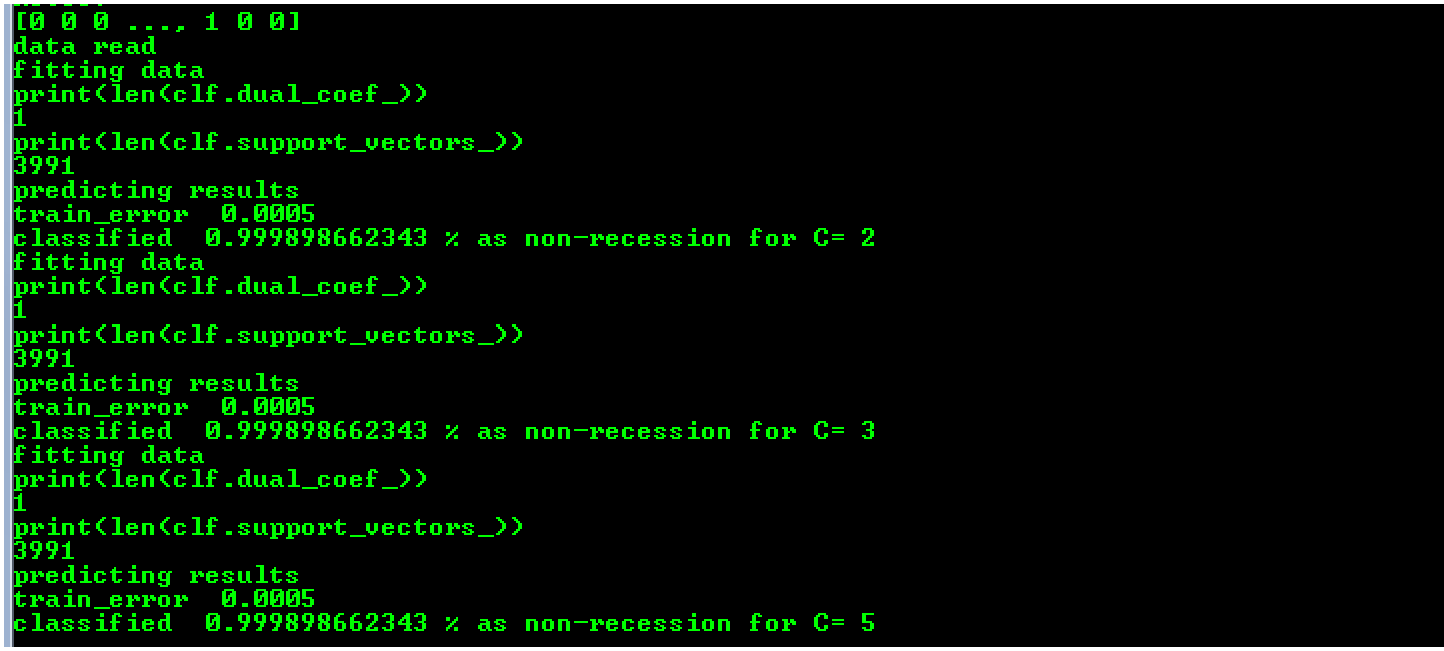
You made the top ten by improving your score by 0.05875.

You just moved up 1 position on the leaderboard.

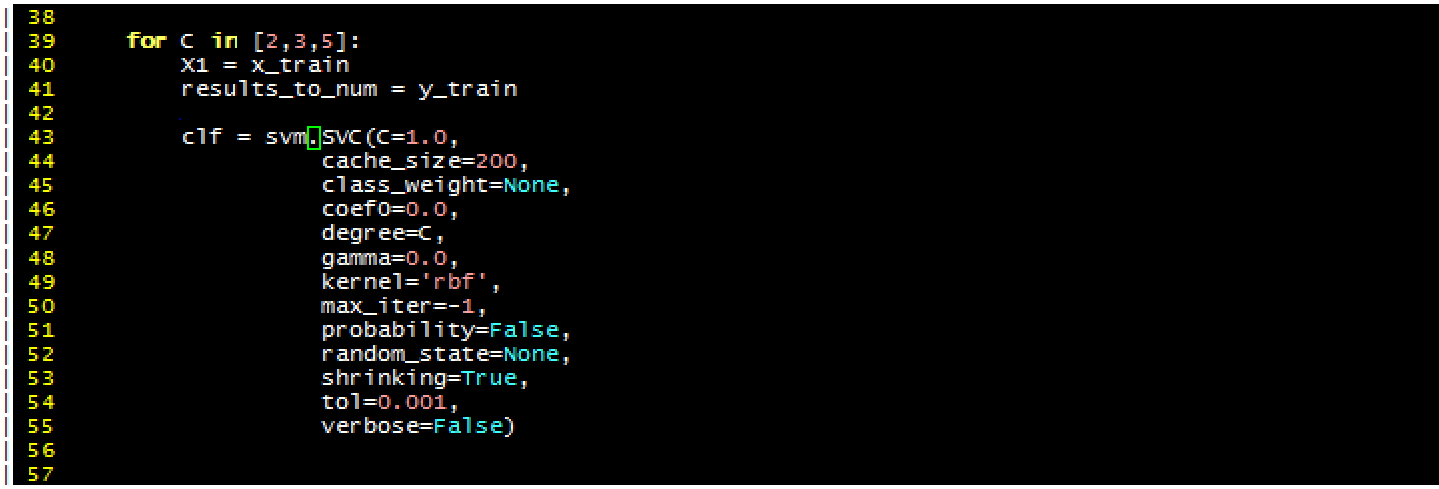
3	new	Eman	0.83600	5	Fri, 06 Feb 2015 00:53:22 (-(

Download raw data

SVM implementation using radial basis functions of varying degrees (2,3,5 deg):



```
[0 0 0 ..., 1 0 0]
data read
fitting data
print(len(clf.dual_coef_))
1
print(len(clf.support_vectors_))
3991
predicting results
train_error  0.0005
classified  0.999898662343 % as non-recession for C= 2
fitting data
print(len(clf.dual_coef_))
1
print(len(clf.support_vectors_))
3991
predicting results
train_error  0.0005
classified  0.999898662343 % as non-recession for C= 3
fitting data
print(len(clf.dual_coef_))
1
print(len(clf.support_vectors_))
3991
predicting results
train_error  0.0005
classified  0.999898662343 % as non-recession for C= 5
```



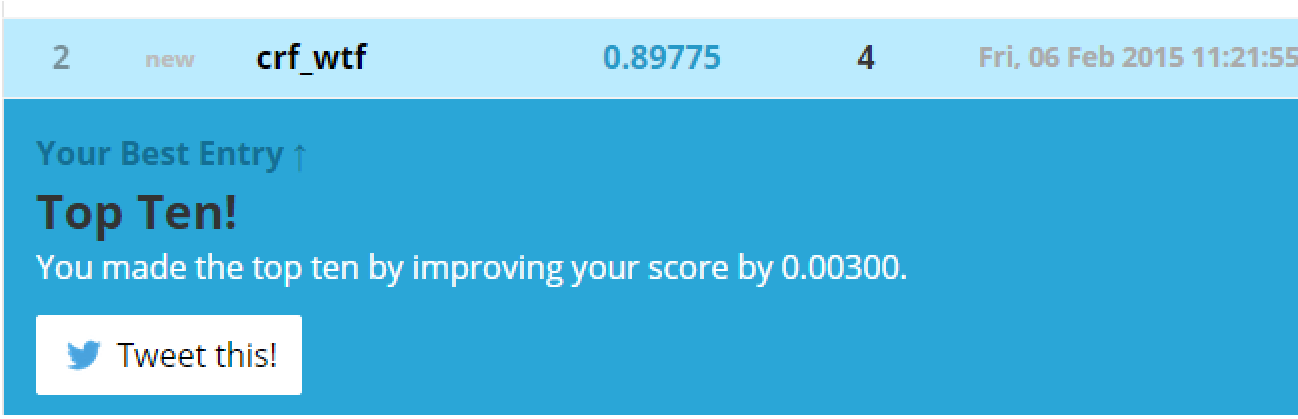
```

    for C in [2,3,5]:
        X1 = x_train
        results_to_num = y_train

        clf = svm.SVC(C=1.0,
                    cache_size=200,
                    class_weight=None,
                    coef0=0.0,
                    degree=C,
                    gamma=0.0,
                    kernel='rbf',
                    max_iter=-1,
                    probability=False,
                    random_state=None,
                    shrinking=True,
                    tol=0.001,
                    verbose=False)


```

This had poor results, leaving most of the points classified as non-recession.

Next tried was Linear Support Vector Classification of varying tolerances (.0001,.001,.01) and it had the various probabilities:

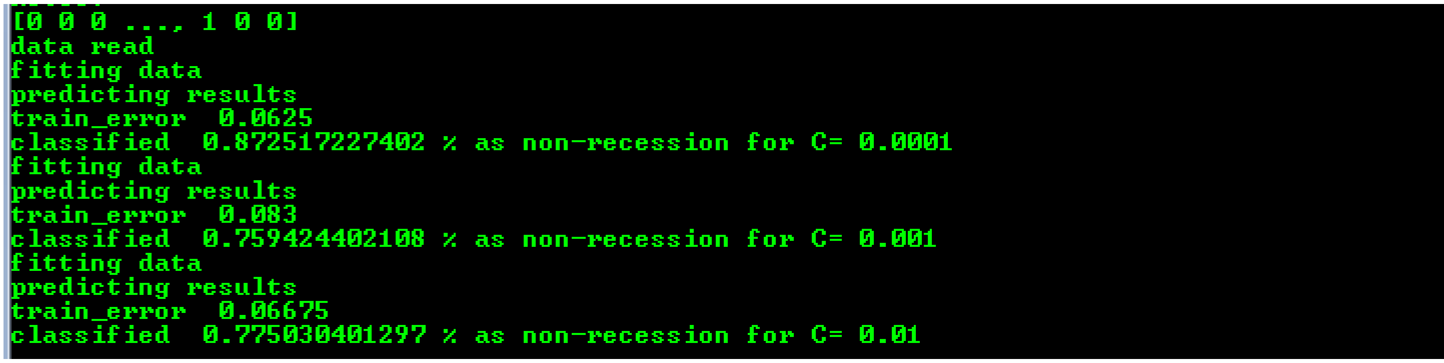
Decided to submit both 0.001 and 0.001:

.0001:



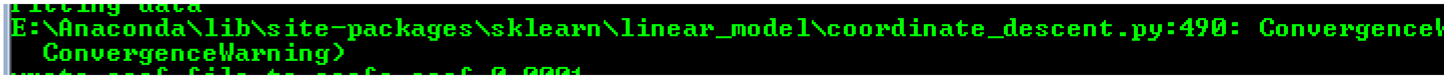
2
new
crf_wtf
0.89775
4
Fri, 06 Feb 2015 11:21:55
Your Best Entry ↑
Top Ten!
You made the top ten by improving your score by 0.00300.
Tweet this!

0.001 only got 86% out of sample error.

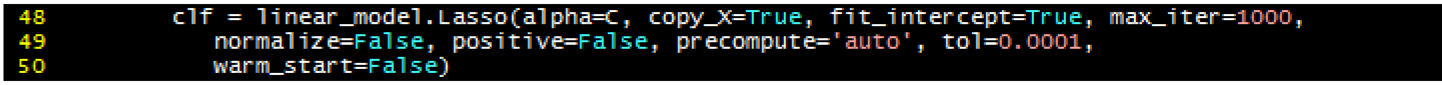


```
[0 0 0 ..., 1 0 0]
data read
fitting data
predicting results
train_error  0.0625
classified  0.872517227402 % as non-recession for C= 0.0001
fitting data
predicting results
train_error  0.083
classified  0.759424402108 % as non-recession for C= 0.001
fitting data
predicting results
train_error  0.06675
classified  0.775030401297 % as non-recession for C= 0.01
```

Try: use lasso to make the weights sparse, then use SVM rbf with kernel trick, lasso has convergence issues on this run:

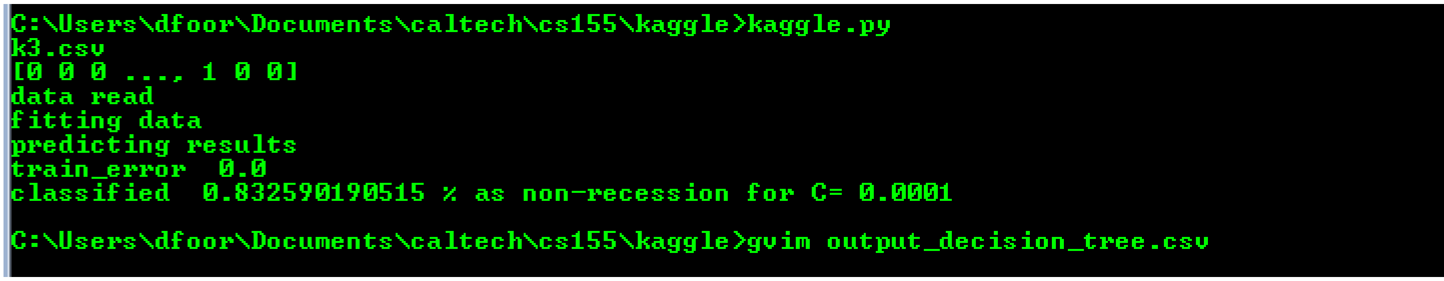


```
fitting data
E:\Anaconda\lib\site-packages\sklearn\linear_model\coordinate_descent.py:490: ConvergenceW
  ConvergenceWarning)
write coef file to coefs coef 0.0001
```



```
        clf = linear_model.Lasso(alpha=C, copy_X=True, fit_intercept=True, max_iter=1000,
           normalize=False, positive=False, precompute='auto', tol=0.0001,
           warm_start=False)
```

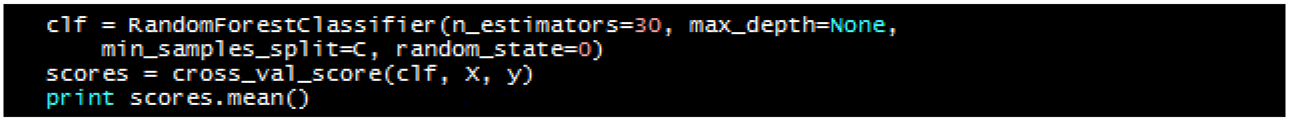
Tryed decision tree and got a very promsing result:



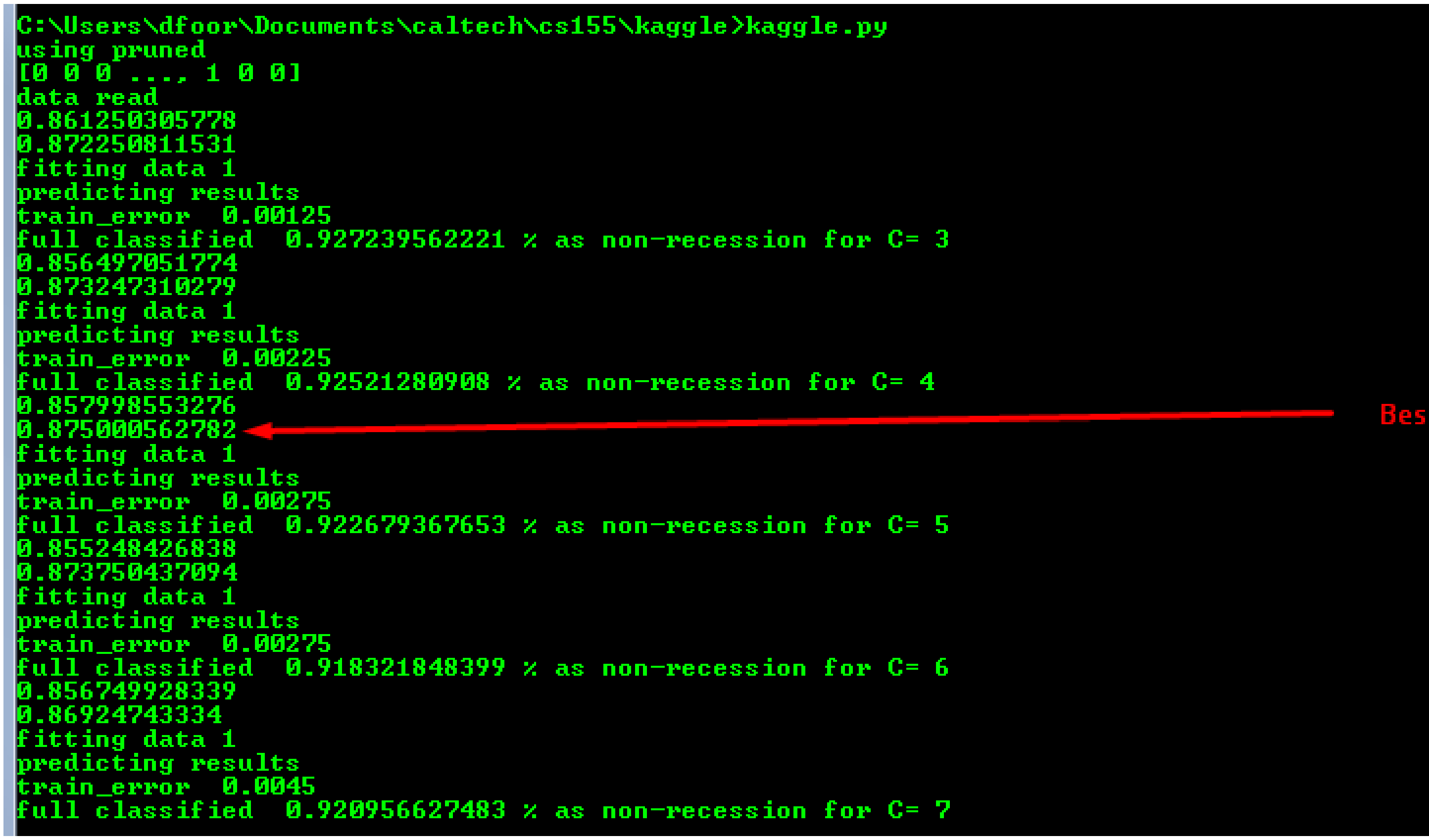
```
C:\Users\dfoor\Documents\caltech\cs155\kaggle>kaggle.py
k3.csv
[0 0 0 ..., 1 0 0]
data read
fitting data
predicting results
train_error  0.0
classified  0.832590190515 % as non-recession for C= 0.0001

C:\Users\dfoor\Documents\caltech\cs155\kaggle>gvim output_decision_tree.csv
```

it classified 83.2 % as belonging to the non-resession, which, is the same distribution as the test set.

Tried random forest using cross validation and selected the model with a minimum group size of 5:

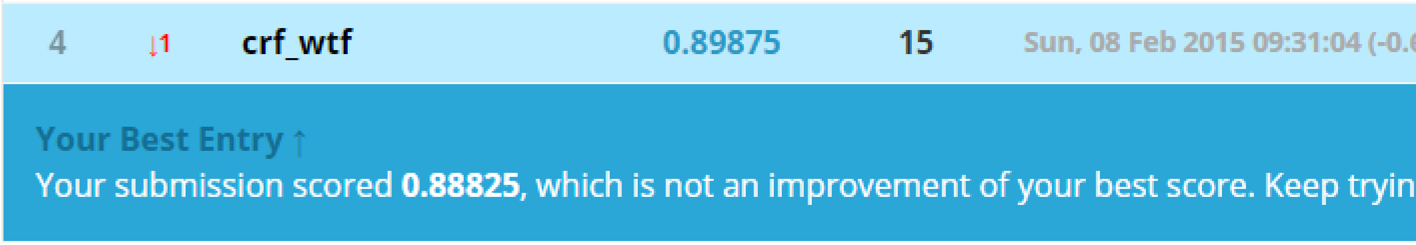


```
  clf = RandomForestClassifier(n_estimators=30, max_depth=None,
      min_samples_split=C, random_state=0)
  scores = cross_val_score(clf, X, y)
  print scores.mean()
```



```
C:\Users\dfoor\Documents\caltech\cs155\kaggle>kaggle.py
using pruned
[0 0 0 ..., 1 0 0]
data read
0.861250305778
0.872250811531
fitting data 1
predicting results
train_error  0.00125
full classified  0.927239562221 % as non-recession for C= 3
0.856497051774
0.873247310279
fitting data 1
predicting results
train_error  0.00225
full classified  0.92521280908 % as non-recession for C= 4
0.857998553276
0.875000562782
fitting data 1
predicting results
train_error  0.00275
full classified  0.922679367653 % as non-recession for C= 5
0.855248426838
0.873750437094
fitting data 1
predicting results
train_error  0.00275
full classified  0.918321848399 % as non-recession for C= 6
0.856749928339
0.86924743334
fitting data 1
predicting results
train_error  0.0045
full classified  0.920956627483 % as non-recession for C= 7
```

Bes



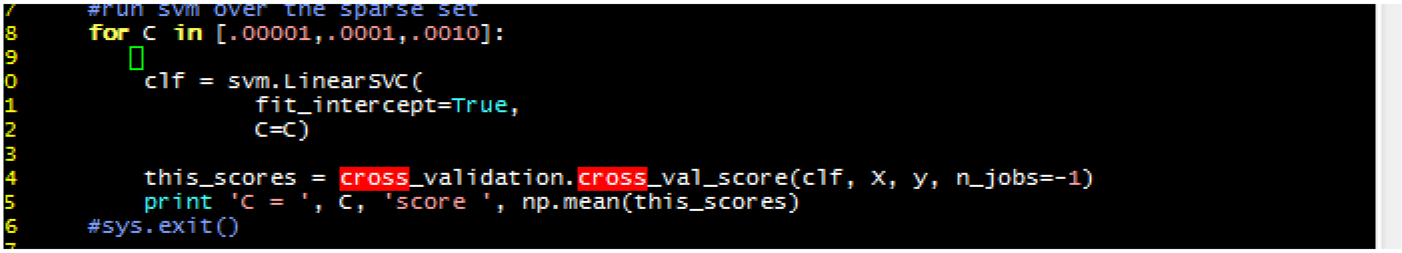
4	↓1	crf_wtf	0.89875	15	Sun, 08 Feb 2015 09:31:04 (-0.(

Your Best Entry ↑

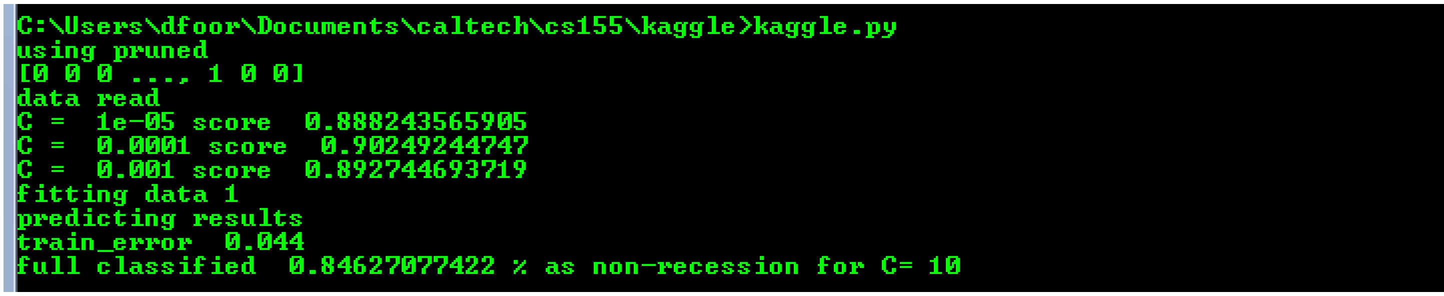
Your submission scored **0.88825**, which is not an improvement of your best score. Keep tryin

Woo hoo. 90%

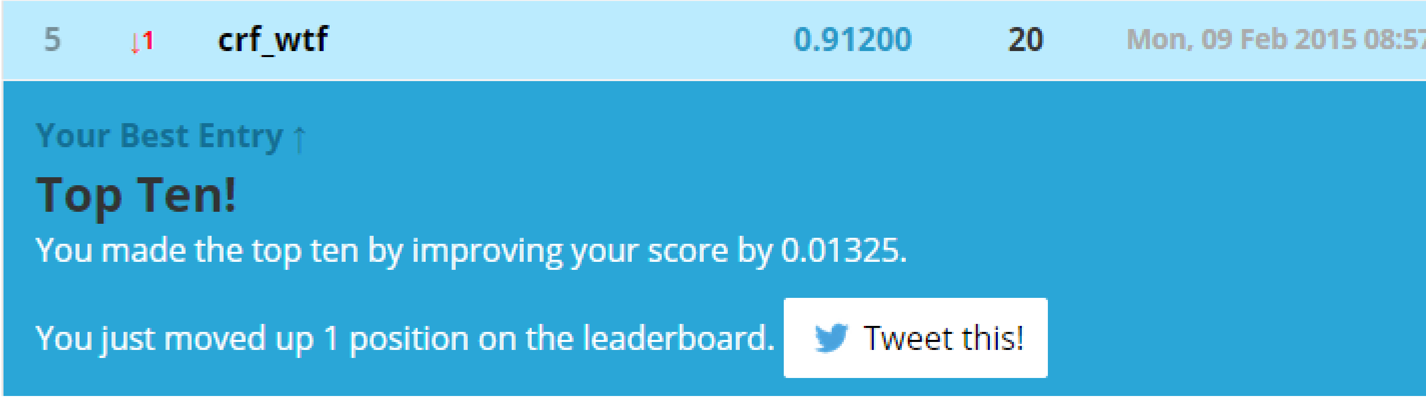
Got smart and started to use crossvalidation. I tried multiple SVM impelentaitons (radial basis functions, stochastic gradient descent on linear model, polynomial models), and found that tuning C to a relaxed .0001 lead to the best performance.



```
    #run svm over the sparse set
    for C in [.00001,.0001,.0010]:
      
        clf = svm.LinearSVC(
                fit_intercept=True,
                C=C)

        this_scores = cross_validation.cross_val_score(clf, X, y, n_jobs=-1)
        print 'C = ', C, 'score ', np.mean(this_scores)
    #sys.exit()
```



```
C:\Users\dfoor\Documents\caltech\cs155\kaggle>kaggle.py
using pruned
[0 0 0 ..., 1 0 0]
data read
C =  1e-05 score  0.888243565905
C =  0.0001 score  0.90249244747
C =  0.001 score  0.892744693719
fitting data 1
predicting results
train_error  0.044
full classified  0.84627077422 % as non-recession for C= 10
```



5 ↓1 crf_wtf 0.91200 20 Mon, 09 Feb 2015 08:57

Your Best Entry ↑

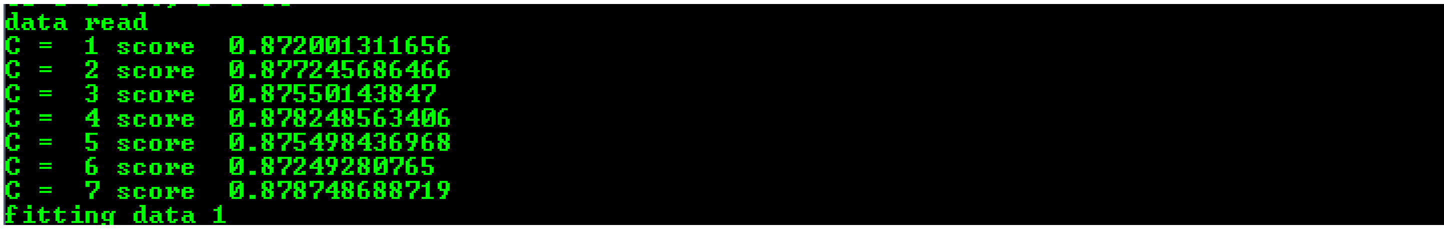
Top Ten!

You made the top ten by improving your score by 0.01325.

You just moved up 1 position on the leaderboard.

Tried random forest classifier in CV:

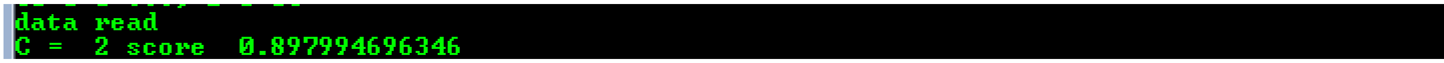
Best was around 88%.



```
data read
C =  1 score  0.872001311656
C =  2 score  0.877245686466
C =  3 score  0.87550143847
C =  4 score  0.878248563406
C =  5 score  0.875498436968
C =  6 score  0.87249280765
C =  7 score  0.878748688719
fitting data 1
```

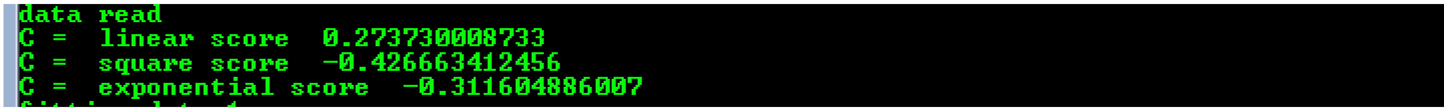
Tried AdaBoostClassiifer:

Best I could get was around 89%:



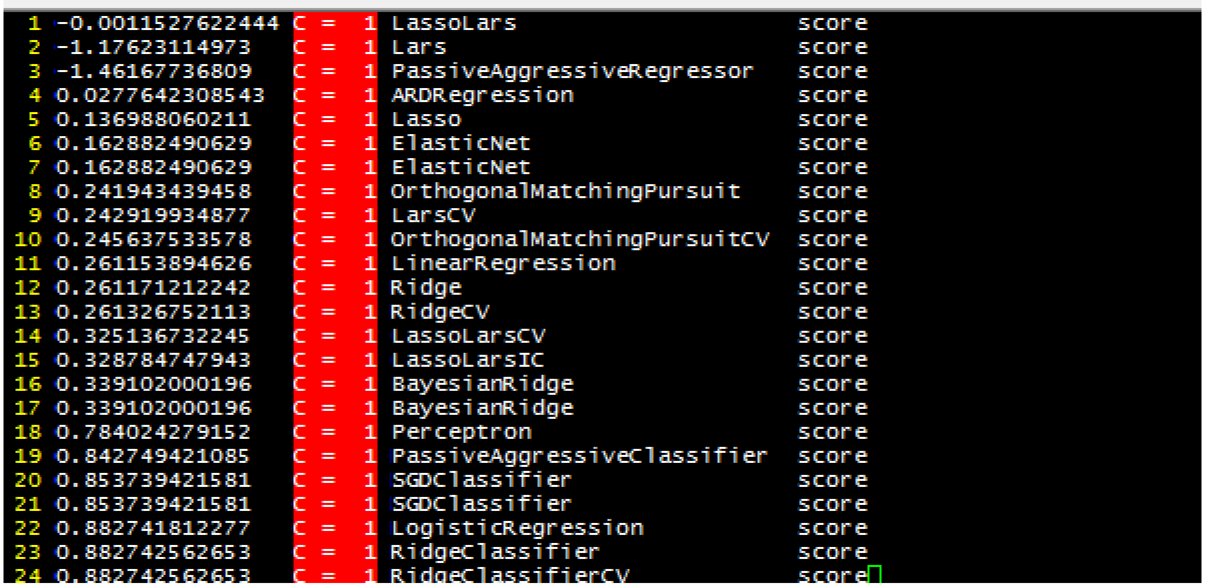
```
data read
C =  2 score  0.897994696346
```

AdaBoostRegressor:



```
data read
C =  linear score  0.273730008733
C =  square score  -0.426663412456
C =  exponential score  -0.311604886007
```

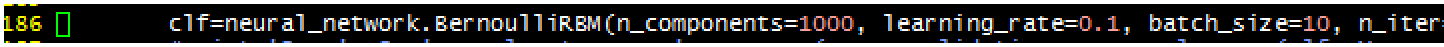
Tried the various linear_model classes. Each that didn't error out are shown with their first-pass CV accuracy. It looks like RidgeClassifier would be a good place to look:



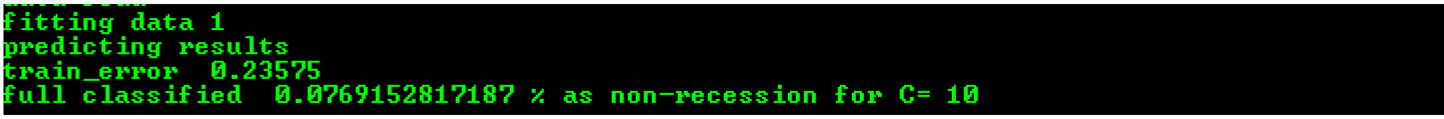
```
-0.0011527622444 C =  1 LassoLars                     score
-1.17623114973   C =  1 Lars                          score
-1.46167736809   C =  1 PassiveAggressiveRegressor    score
0.0277642308543  C =  1 ARDRegression                 score
0.136988060211   C =  1 Lasso                         score
0.162882490629   C =  1 ElasticNet                    score
0.162882490629   C =  1 ElasticNet                    score
0.241943439458   C =  1 OrthogonalMatchingPursuit     score
0.242919934877   C =  1 LarsCV                        score
0.245637533578   C =  1 OrthogonalMatchingPursuitCV   score
0.261153894626   C =  1 LinearRegression              score
0.261171212242   C =  1 Ridge                         score
0.261326752113   C =  1 RidgeCV                       score
0.325136732245   C =  1 LassoLarsCV                   score
0.328784747943   C =  1 LassoLarsIC                   score
0.339102000196   C =  1 BayesianRidge                 score
0.339102000196   C =  1 BayesianRidge                 score
0.784024279152   C =  1 Perceptron                    score
0.842749421085   C =  1 PassiveAggressiveClassifier   score
0.853739421581   C =  1 SGDClassifier                 score
0.853739421581   C =  1 SGDClassifier                 score
0.882741812277   C =  1 LogisticRegression            score
0.882742562653   C =  1 RidgeClassifier               score
0.882742562653   C =  1 RidgeClassifierCV             score
```

Couldn't get Ridge any better.

Tried neural nets;

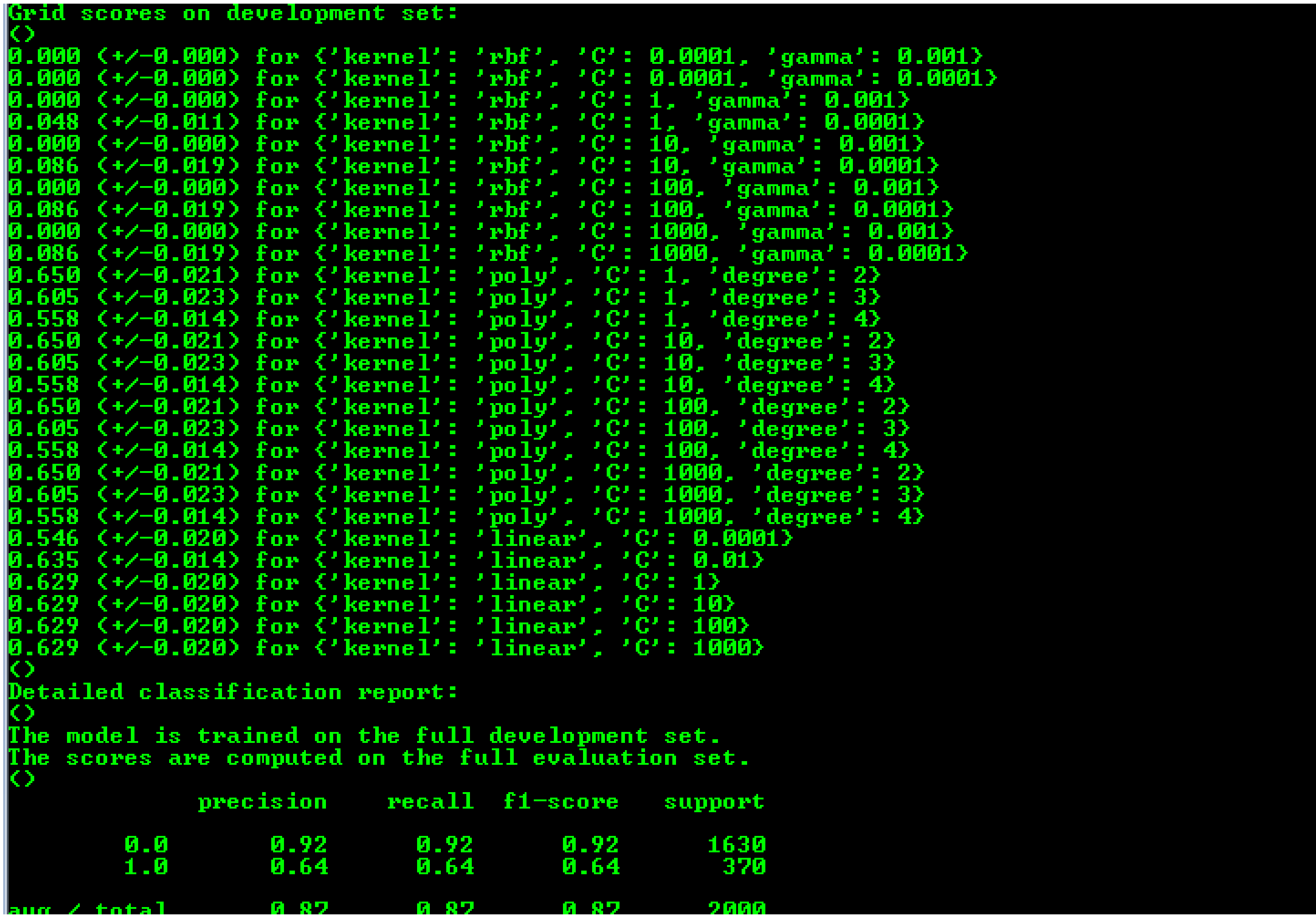


```
        clf=neural_network.BernoulliRBM(n_components=1000, learning_rate=0.1, batch_size=10, n_iter=
```



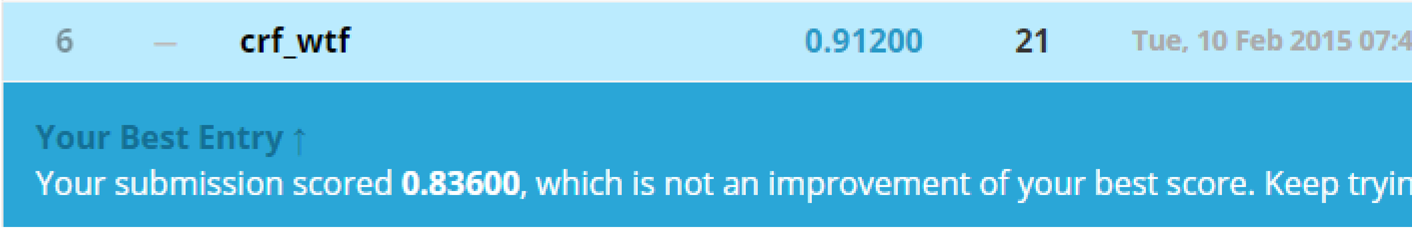
```
fitting data 1
predicting results
train_error  0.23575
full classified  0.0769152817187 % as non-recession for C= 10
```

Used grid search to explore the various parameter spaces. Looks like rbf gets possibly better than the current victor–the linear C=.0001 model.



```
Grid scores on development set:
()
0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 0.0001, 'gamma': 0.001}
0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 0.0001, 'gamma': 0.0001}
0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.048 (+/-0.011) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.086 (+/-0.019) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.086 (+/-0.019) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.086 (+/-0.019) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.650 (+/-0.021) for {'kernel': 'poly', 'C': 1, 'degree': 2}
0.605 (+/-0.023) for {'kernel': 'poly', 'C': 1, 'degree': 3}
0.558 (+/-0.014) for {'kernel': 'poly', 'C': 1, 'degree': 4}
0.650 (+/-0.021) for {'kernel': 'poly', 'C': 10, 'degree': 2}
0.605 (+/-0.023) for {'kernel': 'poly', 'C': 10, 'degree': 3}
0.558 (+/-0.014) for {'kernel': 'poly', 'C': 10, 'degree': 4}
0.650 (+/-0.021) for {'kernel': 'poly', 'C': 100, 'degree': 2}
0.605 (+/-0.023) for {'kernel': 'poly', 'C': 100, 'degree': 3}
0.558 (+/-0.014) for {'kernel': 'poly', 'C': 100, 'degree': 4}
0.650 (+/-0.021) for {'kernel': 'poly', 'C': 1000, 'degree': 2}
0.605 (+/-0.023) for {'kernel': 'poly', 'C': 1000, 'degree': 3}
0.558 (+/-0.014) for {'kernel': 'poly', 'C': 1000, 'degree': 4}
0.546 (+/-0.020) for {'kernel': 'linear', 'C': 0.0001}
0.635 (+/-0.014) for {'kernel': 'linear', 'C': 0.01}
0.629 (+/-0.020) for {'kernel': 'linear', 'C': 1}
0.629 (+/-0.020) for {'kernel': 'linear', 'C': 10}
0.629 (+/-0.020) for {'kernel': 'linear', 'C': 100}
0.629 (+/-0.020) for {'kernel': 'linear', 'C': 1000}
()
Detailed classification report:
()
The model is trained on the full development set.
The scores are computed on the full evaluation set.
()
             precision    recall  f1-score   support

        0.0       0.92      0.92      0.92      1630
        1.0       0.64      0.64      0.64       370

avg / total       0.82      0.82      0.82      2000
```

clearly, nope–and it matches the CV score:



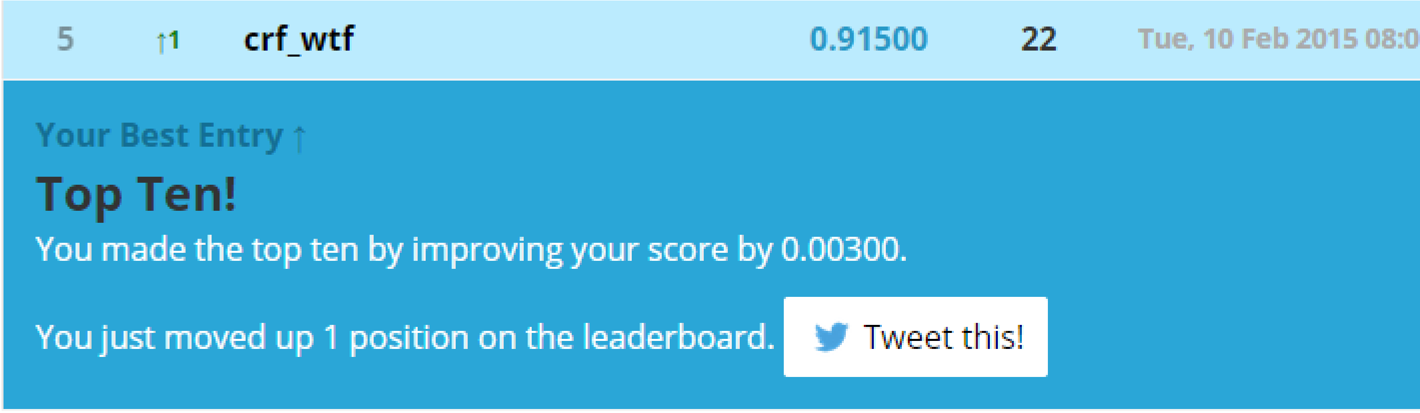
6	—	crf_wtf	0.91200	21	Tue, 10 Feb 2015 07:4

Your Best Entry ↑

Your submission scored **0.83600**, which is not an improvement of your best score. Keep tryin

heh heh.

I took the results from the test set and included them in my training set. Becuase my regularizer is pretty loose, and because I scored better than 1/2 on the test set, i was able to increase the score–i think. I will try this recursivly until something theoretical breaks it.



5	↑1	crf_wtf	0.91500	22	Tue, 10 Feb 2015 08:0

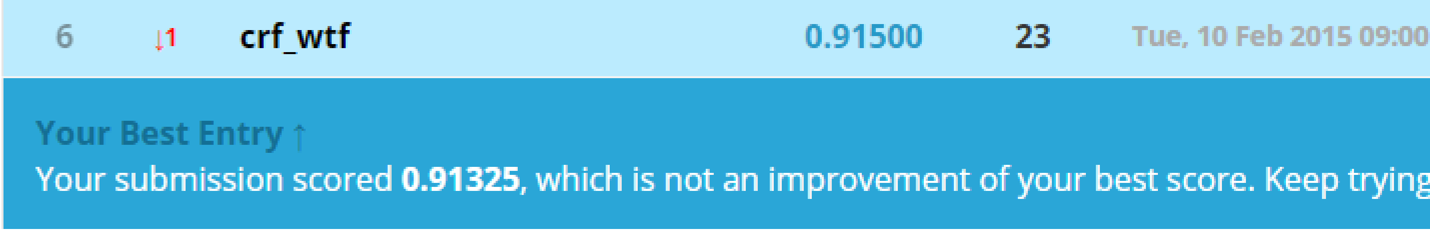
Your Best Entry ↑

Top Ten!

You made the top ten by improving your score by 0.00300.

You just moved up 1 position on the leaderboard.

Second time I tried this, it broke:

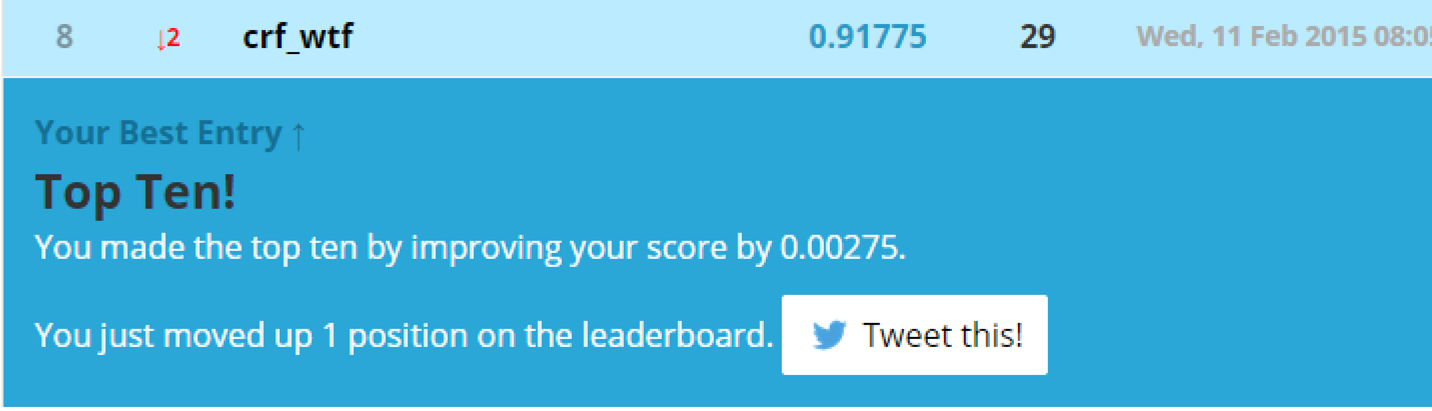


6	↓1	crf_wtf	0.91500	23	Tue, 10 Feb 2015 09:00

Your Best Entry ↑

Your submission scored **0.91325**, which is not an improvement of your best score. Keep trying

Okay–using adaboost, I got a better result and still had training error, so I'll try more classifiers:



8	↓2	crf_wtf	0.91775	29	Wed, 11 Feb 2015 08:0

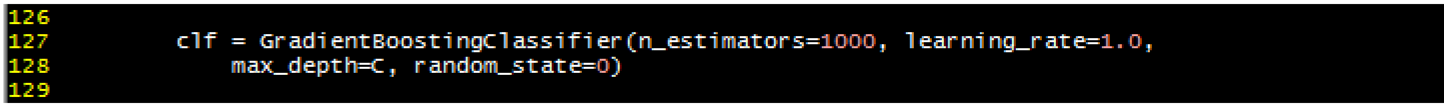
Your Best Entry ↑

Top Ten!

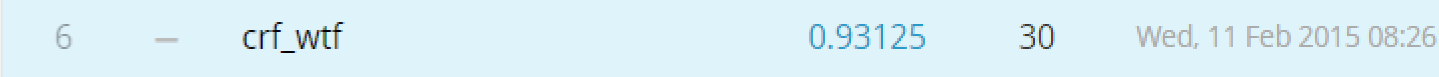
You made the top ten by improving your score by 0.00275.

You just moved up 1 position on the leaderboard.

GradientBoostClassifier returned even better error at 93% and with a tuned max depth of 3:



```
        clf = GradientBoostingClassifier(n_estimators=1000, learning_rate=1.0,
            max_depth=C, random_state=0)
```



6	—	crf_wtf	0.93125	30	Wed, 11 Feb 2015 08:26

5 Conclusion