Great job Mike! After reading your post I had to think for a bit to figure out how one would pull this off and how to generalize to any number of models. As long as you are using linear models it's fairly simple, just takes a bit of geometric thinking. For three models you just need the three 310 length vectors that define the models to be coplanar and 120 degrees apart. Any projection onto three such vectors will sum to exactly 0 and so you will get three models whose predictions will perfectly cancel out each other just like p/1-p. Because spearman correlation isn't a simple linear operation, the average score and thus the total risk doesn't come out to exactly 0 (nor does it for madmax, madmin, and the_guy) but it's close enough! I wrote some code below to illustrate this and made it work for an arbitrary number of models.

import numpy as np import matplotlib.pyplot as plt from scipy.stats import spearmanr

def angle_between(v1, v2): v1_u = v1/ np.linalg.norm(v1) v2_u = v2/ np.linalg.norm(v2) return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))

def planar_vector(q1, q2, angle): return q1*np.cos(angle) + q2*np.sin(angle)

def orthogonalize(v1, v2): e1 = v1 / np.linalg.norm(v1) u2 = v2 - v1*np.dot(v1, v2.T)/np.dot(v1, v1.T) e2 = u2 / np.linalg.norm(u2) return e1, e2

Define our axes. v1 and v2 could be random vectors, fit models, etc., whatever

you think might define a good plane. The first axis is going be be colinear with v1.

v1 = np.random.randn(310) v2 = np.random.randn(310)

Make 90 degrees apart to serve as axes for our plane

axis1,axis2 = orthogonalize(v1,v2)

number_of_models = 3

models = [planar_vector(axis1, axis2, a*np.pi/180.) for a in np.arange(0, 360, 360/number_of_models)]

print(angle_between(models[0],models[1])*180/np.pi) print(angle_between(models[1],models[2])*180/np.pi)

perfs = []

for era in range(20): # Let's simulate live data that changes every week # along with a model that changes every week

```
# generate random data
x = np.random.randn(5000,310)
# generate random model
h = np.random.randn(310)
# get output from random model
y = np.dot(x,h)

# get predictions from co-planar models
preds = [np.dot(x, model) for model in models]

# get correlations for all co-planar models
perfs.append([spearmanr(pred, y).correlation for pred in preds])
```

perfs = np.array(perfs)

plt.plot(perfs) plt.plot(perfs.mean(axis=1)) plt.legend([f'model {i+1}: {perfs[:,i].mean()}' for i in range(number_of_models)] + ['average correlation']) plt.xlabel('era') plt.ylabel('correlation')

[

norisk

790×588 75.2 KB

](https://forum.numer.ai/uploads/default/original/1X/3026915ac2c3fc5cfc7d99d0bed64bce2b2be28e.png)

So you might be thinking: With a lot of models on a plane, it might start getting fairly crowded with the models getting quite

correlated with each other. Is there any way to achieve zero risk but spread the models out in space a bit more? Well in three dimensions, rather than a plane, spreading the model vectors out such that an arbitrary projection will always cancel out doesn't work for an arbitrary number of vectors. I believe the only way to do it exactly is have the models be the vertices of a regular polytope, and so in three dimensions you can only do it for 4,6,8,12, and 20 models/vertices, e.g. the vertices of the Platonic solids. I leave the work of extending the code to 3+ dimensions as an exercise for the reader