




MISSOURI
S&T

CENTER FOR TRANSPORTATION INFRASTRUCTURE AND SAFETY



Adapting Risk Management and Computational Intelligence Network Optimization Techniques to Improve Traffic Throughput and Tail Risk Analysis

by

Donald Wunsch, Ph.D.
Bryce Schumacher

**NUTC
R348**

**A National University Transportation Center
at Missouri University of Science and Technology**

Disclaimer

The contents of this report reflect the views of the author(s), who are responsible for the facts and the accuracy of information presented herein. This document is disseminated under the sponsorship of the Department of Transportation, University Transportation Centers Program and the Center for Transportation Infrastructure and Safety NUTC program at the Missouri University of Science and Technology, in the interest of information exchange. The U.S. Government and Center for Transportation Infrastructure and Safety assumes no liability for the contents or use thereof.

Technical Report Documentation Page

1. Report No. NUTC R348	2. Government Accession No.	3. Recipient's Catalog No.	
4. Title and Subtitle Adapting Risk Management and Computational Intelligence Network Optimization Techniques to Improve Traffic Throughput and Tail Risk Analysis		5. Report Date April 2014	
		6. Performing Organization Code	
7. Author/s Donald Wunsch Bryce Schumacher		8. Performing Organization Report No. Project # 00042533	
9. Performing Organization Name and Address Center for Transportation Infrastructure and Safety/NUTC program Missouri University of Science and Technology 220 Engineering Research Lab Rolla, MO 65409		10. Work Unit No. (TRAIS)	
		11. Contract or Grant No. DTRT06-G-0014	
12. Sponsoring Organization Name and Address U.S. Department of Transportation Research and Innovative Technology Administration 1200 New Jersey Avenue, SE Washington, DC 20590		13. Type of Report and Period Covered Final	
		14. Sponsoring Agency Code	
15. Supplementary Notes			
16. Abstract Risk management techniques are used to analyze fluctuations in uncontrollable variables and keep those fluctuations from impeding the core function of a system or business. Examples of this are making sure that volatility in copper and aluminum prices do not force an aircraft manufacturer to abruptly shut down manufacturing and making sure a failed bank or state does not cause an entire financial system to fail. Computer network optimization techniques involve many nodes and routes communicating to maximize throughput of data while making sure not to deadlock high priority or time sensitive data. This project will involve exploring possible remappings of these application spaces from risk and computer networks to traffic. Some of these possible mappings include mapping flash crashes and black swans to traffic jams, bank failure to construction or traffic accidents, data packets to vehicles, network routers to traffic lights and other intersection policies. Due to the large data and large solution/ state/ policy spaces computational intelligence techniques are a natural fit for traffic as they are for risk management and computer network optimization.			
17. Key Words Risk, optimization, data, finance, traffic, infrastructure	18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.		
19. Security Classification (of this report) unclassified	20. Security Classification (of this page) unclassified	21. No. Of Pages 12	22. Price

Adapting Risk Management and Computational Intelligence Network Optimization Techniques to Improve Traffic Throughput and Tail Risk Analysis

Donald Wunsch, Ph.D.

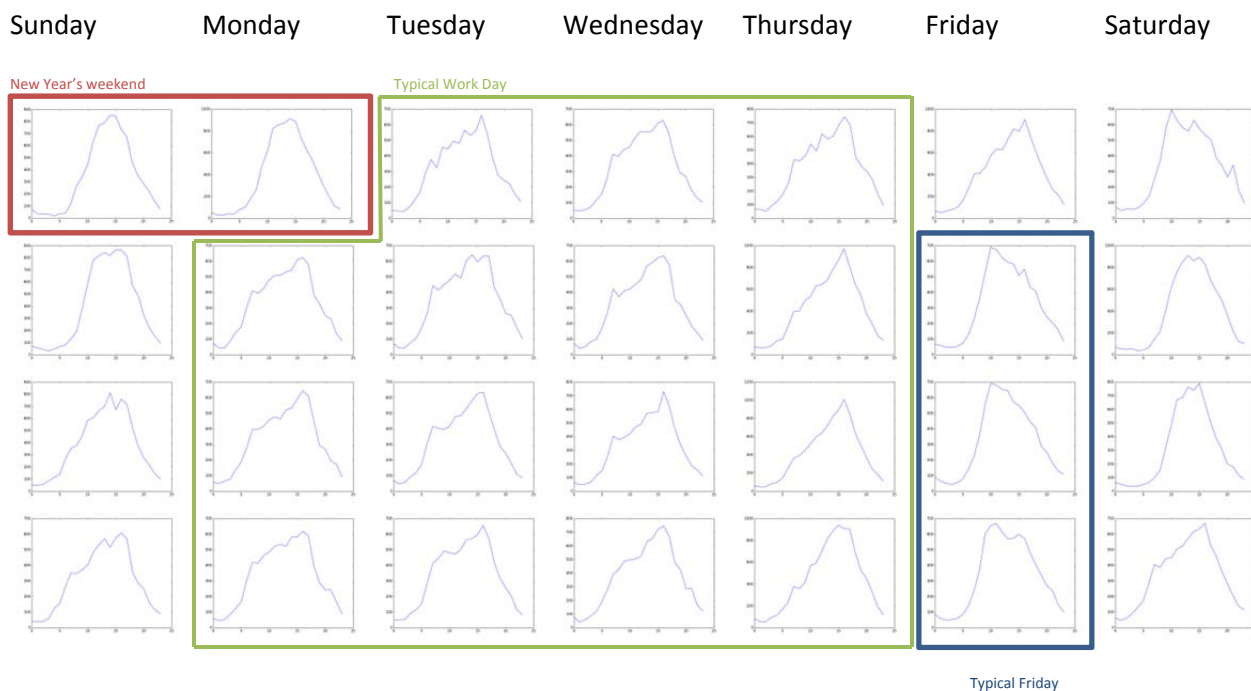
Bryce Schumacher

Data

The data used included 365 days of hourly data in 2011 for training and 366 days of hourly data in 2012 for testing. The data comes from ATR175, a traffic recorder, on an interstate between a metropolitan area and suburb area. The data is an approximation of the number of cars that pass the recorder each hour. This data and other free traffic data can be found at <http://www.dot.state.mn.us/traffic/data/>

Daily Profiles

Below is a sample of hourly data from 28 consecutive days from the 2012 data. Columns correspond to days of the week and rows correspond to consecutive weeks.

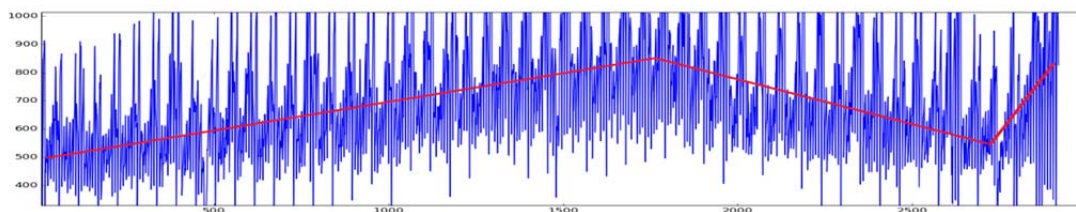


Looking at the traffic plotted for each day above it is possible to quickly identify several common shapes. Typical weekdays highlighted in green show a wide range of morning commute hours but show a large 5 PM peak for

the commute home. Fridays do not have as many cars as regular weekdays (different vertical scale for each graph) and have no 5 PM peak as workers head home early for the weekend. Weekend traffic tends to be tightly banded around mid-day. Mondays after a holiday and Monday holidays are often have a profile similar to a weekend.

Seasonal Trend

The following chart shows the entire 2012 year with low traffic late night and early morning hours removed. Notice the gradual increase from the beginning of the year in traffic. The peak occurs during summer vacation season. There is an additional peak for the winter holidays.



Model Building for Traffic Forecasting

For this investigation the goal was to predict a 24 hour trace of each day. As an example: predicting all 24 hours of tomorrow using only today's traffic counts and any other past data.

Classical decomposition

The classical decomposition as described in [2] is:

$$X_t + m_t + s_t + Y_t$$

Where X_t is the predicted number of cars for an hour, m_t is a trend component, s_t is a seasonal component, and Y_t is a random noise component or non-predictable innovation.

It is possible to build a linear model that is a combination of trend and seasonal components. This is very typical way of doing time series prediction. A model such as this would allow for easy calculation of predictive accuracy and average and standard deviation of prediction errors. This allows for the calculation of confidence intervals and would help in planning for city resources. However it was decided that this model would not be optimal for this application. Using this method a separate model would be needed for each hour of the day. Additionally, both the day-of-the-year and day-of-the-month seasonal components being in piecewise form the component coefficients would resemble a large lookup table where the size of the table would be similar in size to the training data. With each coefficient in the table relying on a very small N samples the regressions that were used to create them would not be reliable. These features of the dataset reduce the impact of the best reasons to use this method; simplicity and transparency/ confidence intervals.

Multilayer Perceptron or similar

A multilayer perceptron neural network or similar can be used to map recent data points, the day of the week, and other information to 24 predictions for the next day. Additionally information about holidays and snow days could be included in the mapping. This type of model is appealing due to ease of deployment and automatic training. Concerns include the black box nature of a multilayer perceptron. Operators would need to put in considerable effort to decode a strange or bad prediction. Also the range of predictions is not known. The network can make a prediction that is dissimilar to anything seen in the past. Lastly the dataset in this investigation is unbalanced in more than one way. Basic batch training of perceptrons included minimizing the error across the entire dataset. Because of the low frequency of holidays and weather related events additional steps and tweaks would need to be made during training.

KNN classifier

A KNN classifier was shown to be a good predictor of 23 hour daily electricity load trace forecasting [1]

A KNN classifier would be a more passive and robust approach for this data. For the implementation in this investigation patterns from the training data are formed by combining adjacent days. Each pattern is 48 hours. For testing, the current 48 hours is compared against the training samples to find the nearest 4 neighbors. Specifically the classifier looks for the four 48 hour patterns that look like the current 48 hours in order to predict 24 hours of tomorrow's traffic. If today is Tuesday March 25th and the model is trying to predict all 24 hours of traffic for tomorrow, Wednesday March 26th then it would combine the 48 hours of Monday 24th and Tuesday March 25th into a vector to compare against the KNN knowledge base. The four nearest neighbors each contribute 24 potential "next day" patterns that are then averaged into the prediction for Wednesday. This provides many benefits over the above methods in transparency and model simplicity. First the prediction cannot be outside what has actually been observed in the past since it is an average of previous days that actually occurred. An operator would be able to determine with minimal effort what is contributing to each prediction. There is no longer 24 separate black or gray box transfer functions from inputs to the 24 hours that are being predicted. Each 24 hour pattern now moves together through the model and each nearest neighbor contributes a continuous 24 hour trace to be averaged into the final model. Longer term seasonal components are also addressed. The incoming pattern will naturally associate with previous patterns that have the same seasonal contribution. The effect of long term non-seasonal trends is reduced as new observed patterns are added to the model. These new patterns already contain the trend component and will also associate appropriately. In this model outliers can now be handled appropriately. Outliers may consist of snow days or holidays occurring on different work days than normal. For example: if it begins to snow during work hours on a Tuesday then the shape of traffic from Monday and Tuesday may match a Wednesday snow fall from a previous year. In this case several things happen. First the previous pattern can be used to predict traffic for the next day's snow day even though they happened at different time during the week. This is because the model is blind to the day of the week and only looks for traffic shape associations. Second the model does not use only one training data outlier to make a prediction. This would be inappropriate as outliers by definition do not have a confidence bound, information/entropy. The outlier will be averaged with three other neighbors to make a more conservative prediction. The outlier helps in the prediction but is appropriately not solely trusted with the task.

When additional data is observed it can seamlessly be added to the KNN model. There is no need for a new training and validation. If the knowledge base becomes too large it can be reduced by known methods without affecting the accuracy significantly.[3]

Results

The KNN predictor with $k=4$ was able to predict traffic with an average error of 55 cars per hour when predicting 24 hour ahead. ATR175 counted 4,088,771 cars for 2012 or on average 467 per hour.

For prediction of number of cars for 5PM (the hour with the most cars) the KNN predictor had an error 112 cars per hour with an average of 906 cars passing the traffic recorder every hour. An MLP neural network, and an $AR(\infty)$ approximation were also used as a comparison. Additionally the two best models; KNN and MLP were combined in an ensemble. The average error per day at 5PM was as follows. Lower is better.

Method	Error per day in number of cars passing the traffic recorder at 5 PM
KNN	112
MLP NN	130
$AR(\infty)$	197
KNN/MLP ensemble	109

Attempts at combining input data from separate traffic recorders (ATR191 and ATR208) were not successful at increasing prediction accuracy. These traffic recorders were also located between urban and suburban areas on interstates used for commuting. It is suspected that the information contained at these recorders correlated too closely with those at ATR175 due to the day of the week, climate, and holidays being identical throughout the city.

Conclusion

The KNN method in this investigation outperformed both the MLP neural network and an autoregressive model. The KNN had a lower prediction error, greater model transparency, and can be updated with new information simply by adding patterns to the knowledge base instead of retraining.

1. Al-Qahtani, Fahad H., Crone, Sven F. Multivariate k-Nearest Neighbor Regression for Time Series data - a novel Algorithm for Forecasting UK Electricity Demand, ISF 2013, Seoul, Korea. http://forecasters.org/wp/wp-content/uploads/gravity_forms/7-2a51b93047891f1ec3608bdbd77ca58d/2013/07/2013-ISF-KNN-for-Time-Series-Data.pdf
2. Brockwell, Peter J. and Davis, Richard A. Introduction to Time Series and Forecasting. New York: Springer, 2002. Print.
3. P. E. Hart, The Condensed Nearest Neighbor Rule. IEEE Transactions on Information Theory 18 (1968) 515–516. doi: 10.1109/TIT.1968.1054155

Code appendix

traffic4.py

```
import csv
import matplotlib.pyplot as plt
import numpy
import operator
import math
import copy
import convert2py

test = []
train = []

with open('ATR175.txt', 'rb') as csvfile:
    testreader = csv.reader(csvfile, delimiter=' ')
    for row in testreader:
        test.append(' ', '.join(row))
for i in range(len(test)):
    test[i] = test[i].split('\t')
for i in range(len(test)):
    for j in range(4, len(test[0])):
        test[i][j] = float(test[i][j])

with open('ATR1752011.txt', 'rb') as csvfile:
    trainreader = csv.reader(csvfile, delimiter=' ')
    for row in trainreader:
        train.append(' ', '.join(row))
for i in range(len(train)):
    train[i] = train[i].split('\t')
for i in range(len(train)):
    for j in range(4, len(train[0])):
        train[i][j] = float(train[i][j])

trainsum = 0
testsum = 0

for i in range(len(train)):
    for j in range(4, len(train[i])):
        trainsum = trainsum + train[i][j]
for i in range(len(test)):
    for j in range(4, len(test[i])):
        testsum = testsum + test[i][j]
print trainsum
print testsum
```



```

difflist = []
diffhour = [0]*24
predictionlist = []
for i in range(1,len(test)-1):
    nndist = [10000, 10000, 10000, 10000]
    nn = [0,0,0,0]
    for j in range(1,len(train)-1):
        dist = 0
        diffm1 = map(operator.sub, test[i-1][4:], train[j-1][4:])
        diffm0 = map(operator.sub, test[i][4:], train[j][4:])
        dist = math.sqrt(sum(k**2 for k in diffm1))
        dist = math.sqrt(sum(k**2 for k in diffm0))

        if dist < nndist[0]:
            nndist[3] = nndist[2]
            nn[3] = nn[2]
            nndist[2] = nndist[1]
            nn[2] = nn[1]
            nndist[1] = nndist[0]
            nn[1] = nn[0]
            nndist[0] = dist
            nn[0] = j
        else:
            if dist < nndist[1]:
                nndist[3] = nndist[2]
                nn[3] = nn[2]
                nndist[2] = nndist[1]
                nn[2] = nn[1]
                nndist[1] = dist
                nn[1] = j
            else:
                if dist < nndist[2]:
                    nndist[3] = nndist[2]
                    nn[3] = nn[2]
                    nndist[2] = dist
                    nn[2] = j
                else:
                    if dist < nndist[3]:
                        nndist[3] = dist
                        nn[3] = j

    # print test[i][:3]
    # print train[nn[0]][:3]
    # print train[nn[0]+1][:3]
    # print nn[1]
    prediction = map(operator.add, train[nn[0]+1][4:], train[nn[1]+1][4:])
    prediction = map(operator.add, prediction, train[nn[2]+1][4:])
    prediction = map(operator.add, prediction, train[nn[3]+1][4:])

for l in range(len(prediction)):

```

```

    prediction[l] = prediction[l]/4
for l in range(4,len(test[i+1])):

    difffhour[l-4] = difffhour[l-4] + abs(test[i+1][l] - prediction[l-4])

diffflist.append(sum(abs(l) for l in map(operator.sub, test[i+1][4:], prediction)))
predictionlist.append(prediction)

# convert2py.convert2py('mlp2.c', [200])
import trained_net

# print test[3][4:]
# tensor = copy.deepcopy(test[1][4:])
# for i in range(len(test[2][4:])):
#     # tensor.append(test[2][4+i])
# print trained_net.trained_net(tensor,[0]*24)

diffvector = []
tensorlist = []
for i in range(len(test)-2):
    tensor = copy.deepcopy(test[i][4:])
    for j in range(len(test[i+1][4:])):
        tensor.append(test[i+1][4+j])

    tensorlist.append(tensor)
    tensor = []

neuraldiff = []
ensemblelist = []
ensemblediff = []
knndiff = []
for i in range(len(test)-2):
    netout = trained_net.trained_net(tensorlist[i], [0]*24)

    ensemble = []
    for j in range(len(predictionlist[i])):
        ensemble.append(numpy.average([predictionlist[i][j], netout[j]]))

    # plt.subplot(3,1,1)
    # plt.plot(range(len(test[i][4:]),test[i+2][4:],range(len(predictionlist[i])),predictionlist[i],range(len(test[i][4:]),netout,
range(len(ensemble)), ensemble)
    # plt.subplot(3,1,2)
    # plt.plot(range(len(predictionlist[i])),predictionlist[i])
    # plt.subplot(3,1,3)
    # plt.plot(range(len(test[i][4:]),trained_net.trained_net(tensor,[0]*24))
    # plt.show()
    tensor = []
    neuraldiff.append(test[i+2][4+16] - netout[16])

```

```

ensemblediff.append(test[i+2][4+16] - ensemble[16])
knndiff.append(test[i+2][4+16] - predictionlist[i][16])

psij = [0.02332, 0.05239, -0.04349, -0.03035, 0.05296, 0.00598, -0.05154, 0.01600, 0.04105, -0.03197, -0.02465, 0.03991, -0.00616, -
0.03953, 0.01082, 0.03209, -.02343, -.01993, .03003, .00591, -.03028, .00718, .02504, -.01711, -.01604, .02255, .00543, -
.02315, .00464, .01950, -.01245, -.01286, .01690, .00485, -.01768, .00290, .01516, -.00902, -.01027, .01265, .00423, -.01348, .00172, .01177, -
.00651, -.00818, .00944, .00363, -.01026, -.00336]

arknn = copy.deepcopy(predictionlist)
for i in range(50, len(predictionlist)):
    arknn[i][16] = arknn[i][16] - 25
    for j in range(len(psi)):
        arknn[i][16] = arknn[i][16] - psi[j]*arknn[i-j-1][16]
arknndiff = []
total=[]
for i in range(len(arknn)):
    arknndiff.append(test[i+2][4+16] - arknn[i][16])
    total.append(test[i][4+16])
plt.plot(range(len(neuraldiff)), neuraldiff, range(len(neuraldiff)), ensemblediff, range(len(neuraldiff)), knndiff)
plt.show()
fileout = open("knndiff.csv", "wb");

for i in range(len(knndiff)):
    fileout.write(str(knndiff[i]))
    fileout.write('\n')

for i in range(len(neuraldiff)):
    neuraldiff[i] = abs(neuraldiff[i])
    ensemblediff[i] = abs(ensemblediff[i])
    knndiff[i] = abs(knndiff[i])
    arknndiff[i] = abs(arknndiff[i])

print 'neural error \t', numpy.average(neuraldiff)
print 'ensemble error \t', numpy.average(ensemblediff)
print 'knn error \t', numpy.average(knndiff)
print 'autoreg error \t', numpy.average(arknndiff)
print 'average \t', numpy.average(total)

```

[convert2py.py](#)

80 lines

See attached

[mlp2.c](#)

800 lines

See attached

trained_net.py

1300 lines

See attached

MBP

<http://mbp.sourceforge.net/>

MLPtest.csv

MLPtrain.csv