Real-time Ranking of Electrical Feeders using Expert Advice*

Hila Becker[1], Marta Arias[2]

[1] Computer Science, Columbia University, New York
[2] Center for Computational Learning Systems, Columbia University, New York

Abstract. We are using machine learning to construct a failure-susceptibility ranking of feeders that supply electricity to the boroughs of New York City. The electricity system is inherently dynamic and driven by environmental conditions and other unpredictable factors, and thus the ability to cope with concept drift in real time is central to our solution. Our approach builds on the ensemble-based notion of learning from expert advice as formulated in the continuous version of the Weighted Majority algorithm [16]. Our method is able to adapt to a changing environment by periodically building and adding new machine learning models (or "experts") based on the latest data, and letting the online learning framework choose what experts to use as predictors based on recent performance. Our system is currently deployed and being tested by New York City's electricity distribution company.

Keywords: Concept Drift, Online Learning, Weighted Majority Algorithm, Ranking

1 Introduction

We are developing a machine learning online system that ranks feeders that supply electricity to the boroughs of New York City according to their susceptibility to impending failure in real-time. Primary feeders constitute a critical part of the distribution grid of New York City; feeder failures put networks, control centers, and field crews under considerable stress, especially during the summer, and cost millions of dollars in Operations and Maintenance expenses annually. Our work is focused on 943 underground primary feeders, distributing electricity to the New York City boroughs of Manhattan, Brooklyn, Queens, and the Bronx. Being able to predict incipient failures in close to real-time could enable crews and operators to take short-term preventative actions thus reducing the risk of failure. More details on this application and an earlier incarnation of our system can be found in [8]. In this paper we present an online machine learning ranking system for the feeders ranking problem that is able to cope with concept drift automatically.

Related Work. The problem of dealing with concept drift in the context of learning from data streams is receiving much attention recently, see e.g. [10, 20, 21, 3, 11, 22, 24, 26]. Most of these algorithms divide the input stream of data into subsets of sequential data (or "data windows"), and repeatedly build predictive models using only one or several contiguous windows of data at a time. These algorithms mainly differ in how a single or a combination of window-specific models are used to make future predictions. We distinguish two broad categories: the ones that maintain a single model and replace it with newer ones to cope with the concept drift [6, 20, 5, 7], and ensemble-based methods that keep a set of models and use combinations of the existing models to make predictions. Ensemble-based algorithms that use averages or weighted averages for future predictions include [21, 22, 24, 3]. All these algorithms are similar in that they use heuristics to estimate the predictive accuracy of the ensemble models and use these to weigh models' predictions. Additionally, the

* This work has been partly supported by a research contract from Consolidated Edison Company of New York

works of Klinkenberg et al. [11, 12, 13, 19] describe and compare several strategies for dealing with concept drift such as selecting base learners adaptively, selecting window size adaptively, selecting examples adaptively, etc.

Our solution falls into the latter category of ensemble-based learners. The main difference is that instead of using heuristics or boosting-like combinations of underlying models we follow the framework of *learning from expert advice*. This framework has been thoroughly studied in the theory community, and offers very strong performance guarantees [16, 2, 23, 4, 1, 15].

Our solution extends the existing algorithms in several ways: (1) it handles concept drift by continually adding new experts to the ensemble, (2) it has been adapted to the problem of ranking, and (3) it uses several ad-hoc parameters to control various aspects of the learning and meta-learning. The algorithm of [14] uses a similar idea of adding and dropping experts throughout the execution of the algorithm but differs from our approach in the type of base learners they use, in the set of tunable parameters, and in the fact that we are performing ranking instead of classification or regression.

2 The Feeder Ranking Problem

Our machine learning ranking system is part of a large-scale research project that is underway at Columbia University in collaboration with Consolidated Edison Company of New York [8]. Here we describe some of the basics so that the reader can understand the context in which the machine learning algorithm is being developed.

Data Inputs. The data available to the system is very diverse, not only in nature but also in location, type, format, etc. A significant amount of work has been devoted to understanding, processing and merging this data into attribute-value vector datasets that can be used by standard machine learning algorithms. Briefly, the main input data sources are:

- **Static Data:** attributes comprising this category are mainly physical characteristics of the feeders such as age and composition of each feeder section as well as attributes reflecting connectivity and topology of the network. These values change rarely; when they do (e.g., some feeder section is replaced or new feeders are added) we manually change the appropriate values.
- **Dynamic Data:** attributes in this category change over time. We distinguish two types based on the nature of the data and its rate of change. *Outage data:* lists all the failures happened starting in 2001 up to date. This data is updated daily and stored in a relational database. *Load-related data:* our system receives measurements of the current load-related attributes of several components of the electricity system. The data needs to be aggregated by feeders in some cases. New data comes in intervals of roughly 20 minutes, accumulating at a rate of several hundred megabytes per day in real-time. These are stored in a relational database.

Machine Learning datasets. Using the available input data, we assemble training and test sets in the following manner:

- **Training datasets:** given a start and end date, we assemble training datasets by using all failures that have occurred between the start and end date as positive examples, and we sample non-failures in the same period of time that serve as negative examples. The attributes that are added for each feeder included in a training set (be it by failure or by sampling) are the static attributes and the most recent dynamic values by the time of failure. These datasets are meant for building ranking models, that is, for building the "experts". [3]

[3] To be precise, we have experimented with alternative ways of assembling datasets but for brevity we will omit an explanation of these in this paper.

– **Test datasets:** test datasets are assembled with respect to a given date and time by obtaining the most recent readings of all dynamic attributes and generating a table that lists for each feeder these values together with the static values.

Ranking, Learning and Evaluating Performance. Whenever new readings of the dynamic attributes are received (in intervals of about 20 minutes), a new test dataset is assembled and all experts' models are applied to this new test dataset. When a ranking model is applied to a test dataset, the result is a ranking of the feeders, that is, a list of all the feeders in the electricity system that are sorted to what the model believes is more likely to fail down to less likely to fail. Hence, we construct one feeder ranking per expert approximately every 20 minutes. The final ranking output by our system is a weighted average of (some of the) experts' rankings (see Section 3 below).

Learning occurs at night: all the failures of the previous day are inspected and the performance of each expert is evaluated based on the rank of the feeder that failed just before each failure. The weights of each expert are updated following the online learning scheme. Additionally, we build new experts periodically and add them to the current ensemble. Poor-scoring experts need to be dropped so that the number of experts does not grow indefinitely. To build new experts, we are using *SVMs* and *MartiRank*[4] on the training datasets as described above.

Let $\mathcal{F} - \{f_1, \ldots, f_k\}$ be a set of feeders that failed on a given date. To evaluate the performance of each expert e_j, we use the *normalized average rank* of the failed feeders, according to the formula

$$\text{PERFORMANCE}(e_j, \mathcal{F}) - 1 - \frac{1}{k}\sum_{i-1}^{k} \frac{rank_j(f_i)}{943}$$

where $rank_j(f)$ represents the rank of the failed feeder f according to feeder ranking by expert e_j just before the time f failed and 943 is the total number of feeders in our rankings. For example, if there are 3 failures ranked 100, 231, and 51, then the corresponding performance is $1 - \frac{1}{3}\left(\frac{100+231+51}{943}\right) \approx 0.865$. Notice that the higher up in the feeder ranking a failure is, the better (higher) the performance is; 0.5 is the expected value if rankings are random. We use the same formula to evaluate the overall system's performance as reported in Section 4.

3 Description of the Algorithm

Our online ranking algorithm is based on the principle of learning from expert advice, and draws on ideas from the Continuous Weighted Majority algorithm [16]. It is in essence a meta-learning approach that predicts by combining the rankings of a set of individual algorithms or "experts". The meta-learning algorithm keeps track of each expert's performance and uses it in determining its contribution to the final prediction. Our set of experts consists of machine learning models that are trained using different data windows from different points in time. Given a list of items, each model predicts a ranking that intends to maximize the area under the ROC curve (AUC) [9]. In the binary case, the items are ranked according to the algorithm's confidence that their label is '1'. To cope with concept drift, new models, trained with the most recent data, are periodically added to the existing ensemble. In order to avoid growing an infinitely large ensemble, models are removed according to a function of their age and performance; the age of a model is the number of days since its creation.

Periodically, we train and add new models to the current ensemble. A parameter f determines how often this happens, i.e., new models will be added every f iterations. When new models are created, we assign each of them a weight to be used as an individual performance measure. We add these models to the ensemble of experts used by the algorithm in making its predictions. The expert ensemble is then presented with a set of items to rank and each expert makes a separate prediction.

[4] *MartiRank* [8] is a ranking algorithm based on the boosting framework in [17].

```
ONLINERANK(M, N, β, a, p, n, f)
 1  θ ← 0
 2  while (true)
 3  do receive example set x
 4     for i ← 1 to θ
 5     do r_i ← RANK(m_i, x)
 6     E ← M top-scoring models according to w_i
 7     predict according to weighted average of w_i × r_i for models in E
 8     receive actual ranking y
 9     for i ← 1 to θ
10     do s_i = PERFORMANCE(m_i, y)
11     s_best, s_worst ← min(s_1, ..., s_θ), max(s_1, ..., s_θ)
12     for i ← 1 to θ
13     do l_i = (s_best − s_i) / (s_best − s_worst)
14        w_i = w_i * β^{l_i}
15     if no new models generated in the last f iterations
16       then train n new models m_{θ+1}, ..., m_{θ+n}
17            w_new ← PERCENTILE(p, {w_1, ..., w_θ})
18            w_{θ+1}, ..., w_{θ+n} ← w_new
19            θ ← θ + n
20            if θ > N
21              then remove the θ − N worst-scoring models according to q_i = w_i * a^{age_i}
22                   θ ← N
```

Fig. 1. Pseudocode for our online ranking algorithm.

The algorithm combines these predictions by ranking the items according to their weighted average rank. It then receives the true ranking of the items and updates the weights of the experts in the ensemble. The weight update function is similar to the one discussed in [16], where the weight is multiplied by a function of the loss. The loss of each expert in the ensemble is a measure of its performance, relative to the other experts.

There are several input parameters that can be used to tune the performance of the algorithm. The *learning rate* $\beta \in [0, 1)$ is used in the weight update function to adjust the fraction by which the weights are reduced. A larger value of β corresponds to a slower learning rate, making the algorithm more forgiving to experts that make a mistake by reducing their influence by a smaller fraction. The *budget* N determines the number of models that the algorithm can keep track of at each iteration. Since we do not use a static set of experts as in the traditional weighted majority approach, we have to make sure that our ensemble does not grow infinitely when we add new models. We can also adjust the number of models that the algorithm uses for prediction. In the traditional approach, the advice of all experts in the ensemble is combined to make the final prediction. By using a parameter M for the number of predictors, we can try to further enhance the performance, combining only the advice of top performing experts.

Since we add and remove models from our expert ensemble throughout the algorithm, additional parameters are introduced. Let n be the number of new models added to the ensemble. This parameter n depends on how many machine learning algorithms we use (currently two) and on how many training sets we assemble (we vary the training data windows, currently set to 7 and 14 days). When these new models are added, they are assigned an initial weight w_{new}. This weight can be also adjusted to reflect our trust in these new models, and should be relative to the weights of the existing models in the ensemble. We use a parameter p that determines what weight to assign new models as a percentile in the range $[w_{min}, w_{max}]$ for the minimum and maximum weights of the existing models. We also need to decide what models to drop when the ensemble size grows larger

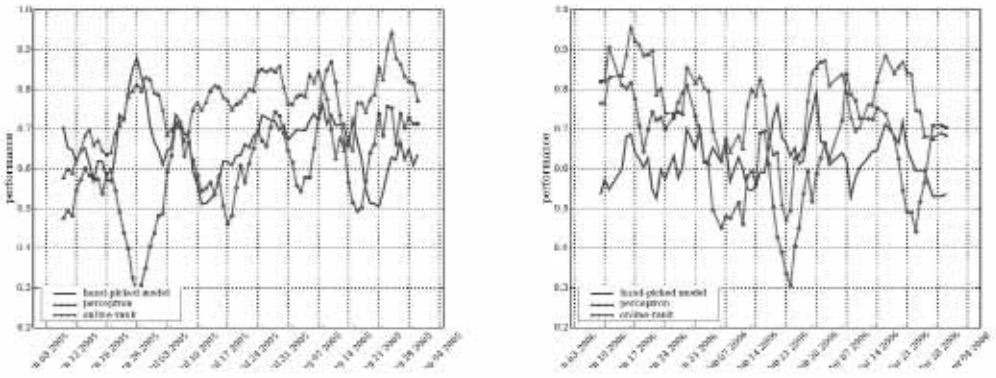


Fig. 2. OnlineRank performance over baseline methods for Summer 2006 (left) and Winter 2006 (right).

than N. We order the experts according to a function of their performance and age, where a is a parameter used to set the exponential decay by age. Pseudocode of our online ranking algorithm can be found in Figure 1.

4 Experimental Results

In this section, we present various experiments with the goal of studying and evaluating the online ranking system. The data used for these experiments was collected between June 2005 and December 2006. The default parameters we use are learning rate $\beta = 0.9$, budget $N = 50$, ensemble size $M = 10$, new model frequency $f = 7$, age decay $a = 0.99$, and new weight percentile $p = 0.7$. For convenience, some of the results use slightly different parameters, we state this where it applies.

We look at the performance of the system over time by plotting the normalized average rank of failures per day on a continuous timeline as explained in Section 2.

We compare our results to those of two separate baseline experiments. The first one uses a ranking version of the Perceptron algorithm [18]. Since we are interested in generating a ranking, we must test the Perceptron model on each one of our feeders and sort them according to their distance from the boundary in descending order. The second baseline comparison involves using a single model throughout the whole run. This model has been hand-tuned by experts who have an insight into the behavior of the system and can estimate which attributes carry more weight toward the final outcome. The performance of these baseline approaches can be seen alongside our online ranking method in Figure 2 for the summer of 2005 and for the winter of 2006. We can see that for both the summer and the winter months our online ranking approach outperforms both the Perceptron and the fixed model. In this experiment we used ensemble size $M = 50$.

We examine the effects of varying the value of a single parameter while keeping the rest constant. Figure 3 shows the performance of the system during the summer of 2006 for different values of experts N, which corresponds to the maximum number of experts that the algorithm can select from to make its prediction. We can observe that the performance of the system is directly correlated with the number of existing experts. Intuitively, when there is a larger pool of models to select from, we have a higher chance of selecting the top performing models amongst them, especially if the predicting ensemble size M is small.

Another parameter that we are interested in observing is the new models' weight percentile p, which determines our degree of belief in the incoming models. The performance of the online ranking system with varying weight percentile during the summer of 2006 can be seen in Figure 3. Assigning new models the lowest weight in the range is understandably a poor choice since new

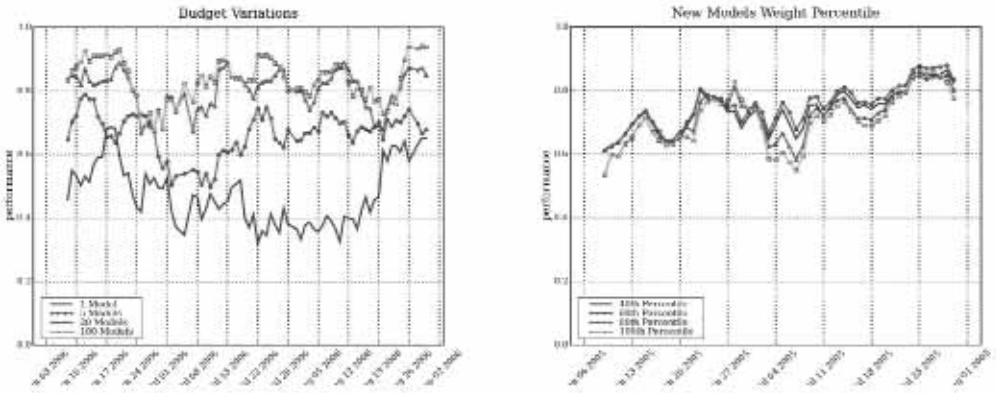


Fig. 3. Online system performance with varying budget N during Summer 2006, here $M-1$ (left) and varying new weight percentile p during Summer 2005 (right).

models are trained with the most recent data, thus carrying an up to date information about the system. On the other hand, assigning too high a weight may force the system to use the newest models always, which may not be a good choice if an older model has been found to work well. We find counterintuitive that the variation in performance in this experiment is so small; we are in the process of trying to figure out why this is so through more experimentation.

5 Conclusions and Future Work

In this paper we have presented an online machine learning system being developed for the problem of ranking feeders that provide electricity to New York City according to their likelihood to impending failure. Our inital results are encouraging and we are in the process of further developing and evaluating our system in terms of optimal parameter tuning, both for this particular application as well as under different types of concept drift. We are planning to include an engine for the detection of concept drift, so that instead of periodically adding new experts we will only add them when a change is detected [6]. We also want to control the diversity of the ensemble, which has been found to improve predictive performance [22]. Finally, we are investigating including a meta-learning layer that would exploit results seen in previous years, e.g. if a model was successful in the previous summer but was retired during the winter, it should be rescued back in the upcoming summer if similar environmental conditions reappear. Examples of other systems that exploit re-occurring contexts are [25, 26].

Acknowledgments. We gratefully acknowledge the support provided by Consolidated Edison Company of New York. We wish to thank all of those who gave so generously in time, advice, and effort. Additionally, a number of Columbia faculty, students, staff, and researchers contributed towards this effort: Phil Gross, Albert Boulanger, Chris Murphy, Luis Alonso, Joey Fortuna, Gail Kaiser, Roger Anderson, and Dave Waltz.

References

1. Olivier Bousquet and Manfred K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363 396, 2002.
2. Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *J. ACM*, 44(3):427–485, 1997.

3. Fang Chu and Carlo Zaniolo. Fast and light boosting for adaptive mining of data streams. In *Proceedings of the Pacific-Asia Knowledge Discovery and Data Mining Conference*, pages 282–292, 2004.
4. Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *STOC*, pages 334–343, 1997.
5. João Gama and Gladys Castillo. Learning with local drift detection. In *ADMA*, pages 42–55, 2006.
6. João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. Learning with drift detection. In *SBIA*, pages 286–295, 2004.
7. João Gama, Pedro Medas, and Pedro Pereira Rodrigues. Learning decision trees from dynamic data streams. In *SAC*, pages 573–577, 2005.
8. Philip Gross, Albert Boulanger, Marta Arias, David L. Waltz, Philip M. Long, Charles Lawson, Roger Anderson, Matthew Koenig, Mark Mastrocinque, William Fairechio, John A. Johnson, Serena Lee, Frank Doherty, and Arthur Kressner. Predicting electricity distribution feeder failures using machine learning susceptibility analysis. In *AAAI*, 2006.
9. J.A. Hanley and B.J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 146(1):29–36, 1982.
10. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.
11. Ralf Klinkenberg. Meta-learning, model selection, and example selection in machine learning domains with concept drift. In *LWA*, pages 164–171, 2005.
12. Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In Pat Langley, editor, *Proceedings of ICML-00, 17th International Conference on Machine Learning*, pages 487–494, Stanford, US, 2000. Morgan Kaufmann Publishers, San Francisco, US.
13. Ralf Klinkenberg and Stefan Rüping. Concept drift and the importance of example. In *Text Mining*, pages 55–78. Jürgen Franke, Gholamreza Nakhaeizadeh, and Ingrid Renz, 2003.
14. Jeremy Z. Kolter and Marcus A. Maloof. Using additive expert ensembles to cope with concept drift. In *ICML*, pages 449–456, 2005.
15. Yi Li and Philip M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1-3):361–387, 2002.
16. Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.
17. Philip M. Long and Rocco A. Servedio. Martingale boosting. In *COLT*, pages 79–94, 2005.
18. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(1):386–407, 1958.
19. Martin Scholz and Ralf Klinkenberg. An ensemble classifier for drifting concepts. In *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*, pages 53–64, 2005.
20. Milton Severo and João Gama. Change detection with kalman filter and cusum. In *Discovery Science*, pages 243–254, 2006.
21. K. Stanley. Learning concept drift with a committee of decision trees, 2001.
22. W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382, 2001.
23. V. G. Vovk. Aggregating strategies. In *Proceedings of the Conference on Computational Learning Theory*, pages 371–386, 1990.
24. Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, 2003.
25. Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.
26. Ying Yang, Xindong Wu, and Xingquan Zhu. Combining proactive and reactive predictions for data streams. *Data Mining and Knowledge Discovery*, 13:261–289, 2006.

Real-time Ranking of Electrical Feeders using Expert Advice

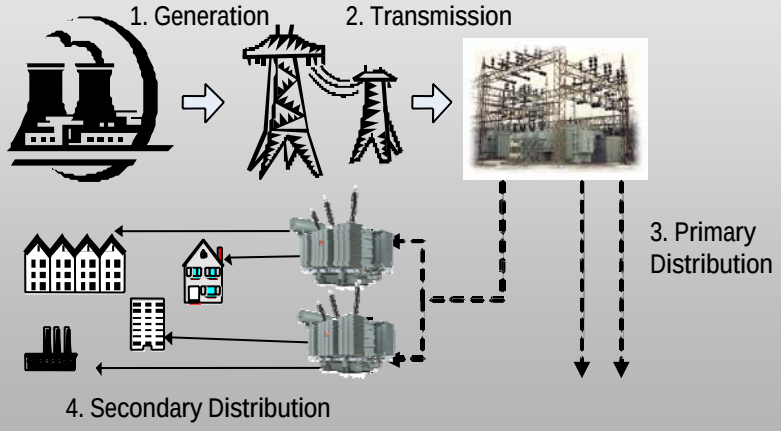
Hila Becker [1,2], Marta Arias [1]
[1]Center for Computational Learning Systems
[2]Computer Science Department
Columbia University

Overview

- The problem
  - The electrical system
  - Available data
- Approach
- Challenges
- Our solution using Online learning
- Experimental results

2

The Electrical System



3

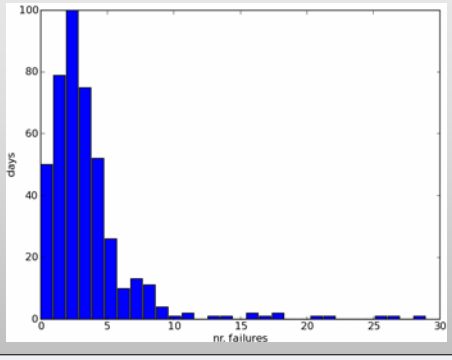
The Problem

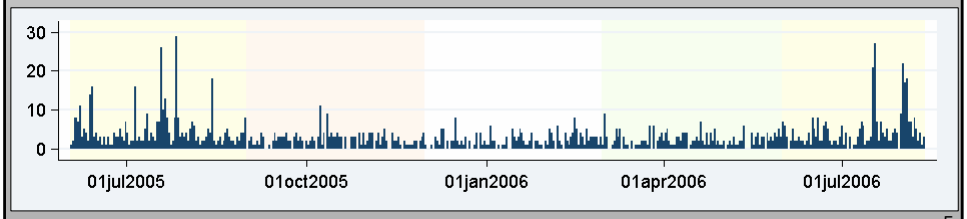
- Distribution feeder failures result in automatic feeder **shutdown**
  - called “Open Autos” or O/As
- O/As **stress** networks, control centers, and field crews
- O/As are **expensive** ($ millions annually)
- Proactive replacement is much cheaper and safer than reactive repair
- How do we know which feeders to fix?

4

Some facts about feeders and failures

- mostly 0-5 failures per day
- more in the summer
- strong seasonality effects





5

Feeder data

- **Static** data
  - Compositional/structural
  - Electrical
- **Dynamic** data
  - Outage history (updated daily)
  - Load measurements (updated every 15 minutes)
- Roughly **200** attributes for each feeder
  - New ones are still being added.

6

Overview

- The problem
  - The electrical system
  - Available data
- **Approach**
- Challenges
- Our solution using Online learning
- Experimental results

7

Machine Learning Approach

- Leverage Con Edison's domain knowledge and resources
- Learn to rank feeders based on failure susceptibility
- How?
  - Assemble data
  - Train ranking model based on past data
  - Re-rank frequently using model on current data

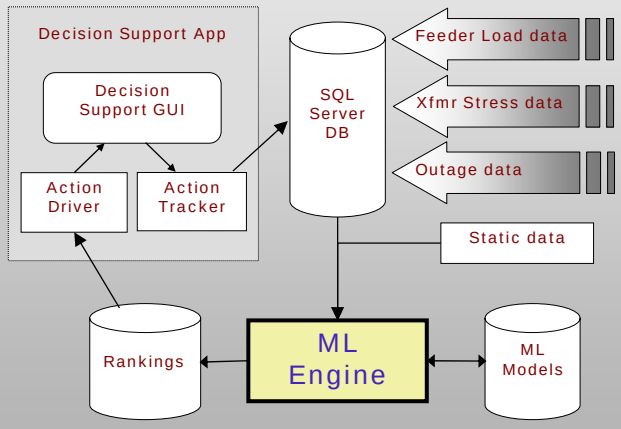
8

Feeder Ranking Application

- Goal: **rank** feeders according to failure susceptibility
  - High risk placed near the top
- Integrate different types of data
- Interface that reflects the latest state of the system
  - Update feeder ranking every 15 min.

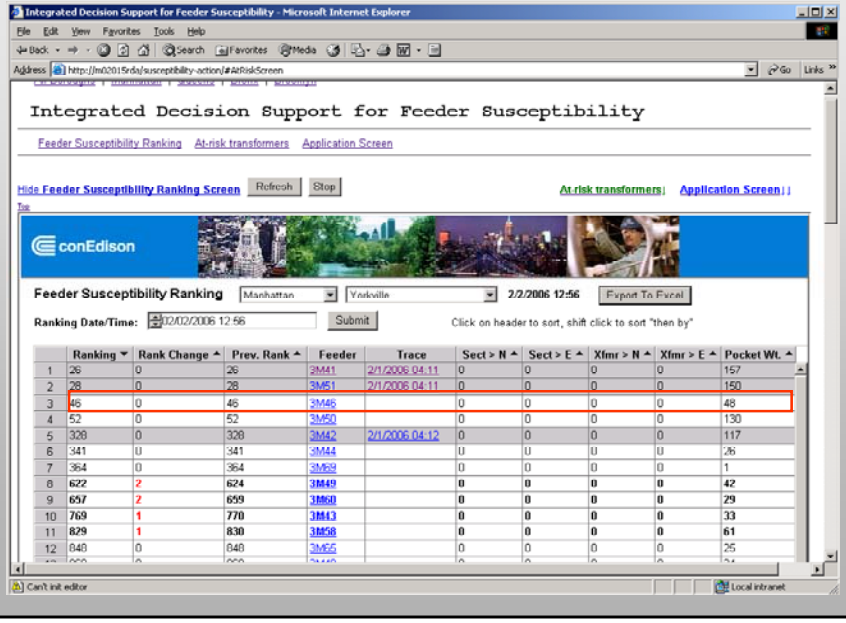
9

Application Structure



10

Decision Support GUI



11

Overview

- The problem
  - The electrical system
  - Available data
- Approach
- **Challenges**
- Our solution using Online learning
- Experimental results

12

Simple Solution

- Supervised batch-learning algorithms
  - Use past data to train a model
  - Re-rank frequently using this model on current data
- Use the best performing learning algorithm
  - How do we measure performance?
  - MartiRank - boosting algorithm by [Long & Servedio, 2005]
    - Use MartiRank for dynamic feeder ranking

13

Performance Metric

- Normalized **average rank of failed feeders**

$$1 - \frac{\sum_i rank(failure_i)}{\#failures * \#feeders}$$

14

Performance Metric Example

$$1 - \frac{\sum_i rank(failure_i)}{\#failures * \#feeders}$$

$$1 - \frac{2+3+5}{3*8} = 0.5833$$

ranking	outages
1	0
2	1
3	1
4	0
5	1
6	0
7	0
8	0

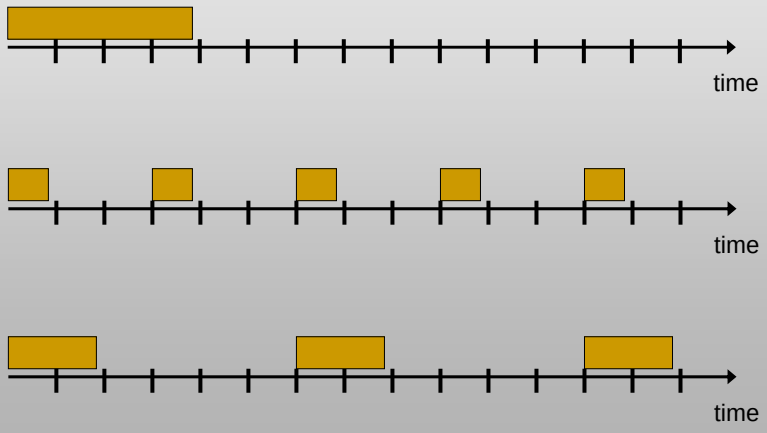
15

Real-time ranking with MartiRank

- MartiRank is a "**batch**" learning algorithm
- Deal with changing system by:
  - generating new datasets with latest data
  - Re-training new model, replacing old model
  - Using newest model to generate ranking
- Must implement "training strategies"

16

Real-time ranking with MartiRank



17

How to measure performance over time

- Every ~15 minutes, generate new ranking based on current model and latest data
- Whenever there is a failure, look up its rank in the latest ranking before the failure
- After a whole day, compute normalized average rank

18

Using MartiRank for real-time ranking of feeders

- MartiRank seems to work well, but..
  - User decides when to re-train
  - User decides how much data to use for re-training
  - Performance degrades over time
- Want to make system automatic
  - Do not discard all old models
  - Let the system decide which models to use

19

Overview

- The problem
  - The electrical system
  - Available data
- Approach
- Challenges
- **Our solution using Online learning**
- Experimental results

20

Learning from expert advice

- Consider each model as an expert
- Each expert has associated weight
  - Reward/penalize experts with good/bad predictions
  - Weight is a measure of confidence in expert's prediction
- Predict using weighted average of top-scoring experts

21

Learning from expert advice

- Advantages
  - Fully automatic
  - Adaptive
  - Can use many types of underlying learning algorithms
  - Good performance guarantees from learning theory: *performance never too far off from best expert in hindsight*
- Disadvantages
  - Computational cost: need to track many models "in parallel"

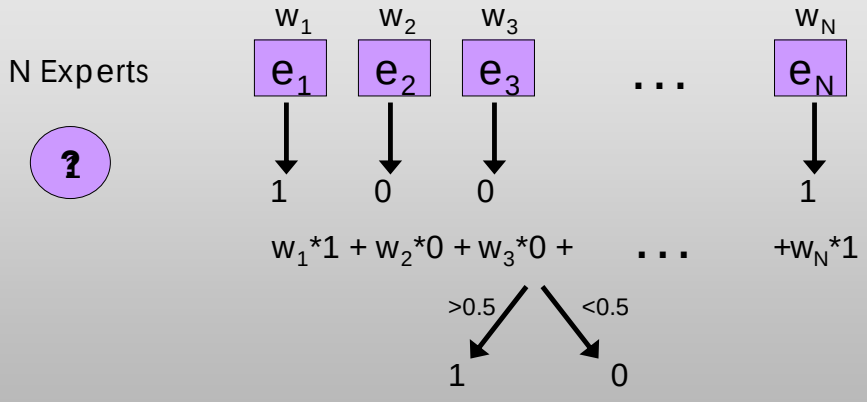
22

Weighted Majority Algorithm [Littlestone & Warmuth '88]

- Introduced for binary classification
  - Experts make predictions in [0,1]
  - Obtain losses in [0,1]
- Pseudocode:
  - Learning rate as main parameter, ß in (0,1]
  - There are N "experts", initially weight is 1 for all
  - For t=1,2,3, …
    - Predict using weighted average of experts' prediction
    - Obtain "true" label; each expert incurs loss l_i
    - Update experts' weights using $w_{i,t+1} = w_{i,t} \cdot pow(ß,l_i)$

23

Weighted Majority Algorithm



- Calculate li = loss(i) for i=1,…,N
- Update: wi,t+1 = wi,t • pow(ß,li) for learning rate ß, time t and i=1,..,N

24

In our case, can't use WM directly

- Use ranking as opposed to binary classification
- More importantly, do not have a fixed set of experts

25

Dealing with ranking vs. binary classification

- Ranking loss as normalized average rank of failures as seen before, loss in [0,1]
- To combine rankings, use a weighted average of feeders' ranks

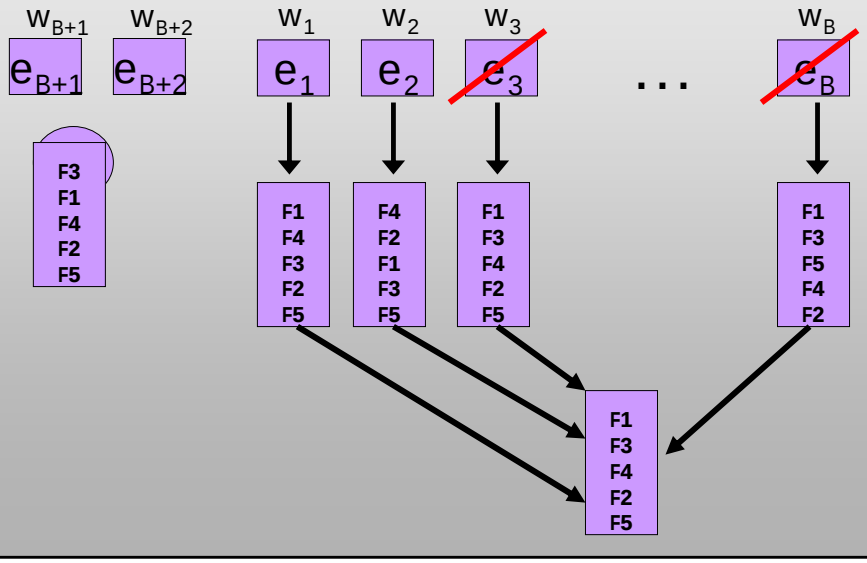
26

Dealing with a moving set of experts

- Introduce new parameters
  - B: "budget" (max number of models) set to 100
  - p: new models weight percentile in [0,100]
  - α: age penalty in (0,1]
- If too many models (more than B), drop models with poor q-score, where
  - $q_i = w_i \bullet pow(\alpha, age_i)$
  - I.e., α is rate of exponential decay

27

Online Ranking Algorithm



28

Other parameters

- How often do we train and add new models?
  - Hand-tuned over the course of the summer
  - Alternatively, one could train when observed performance drops .. not used yet
- How much data do we use to train models?
  - Based on observed performance and early experiments
    - 1 week worth of data, and
    - 2 weeks worth of data

29

Overview

- The problem
  - The electrical system
  - Available data
- Approach
- Challenges
- Our solution using Online learning
- **Experimental results**

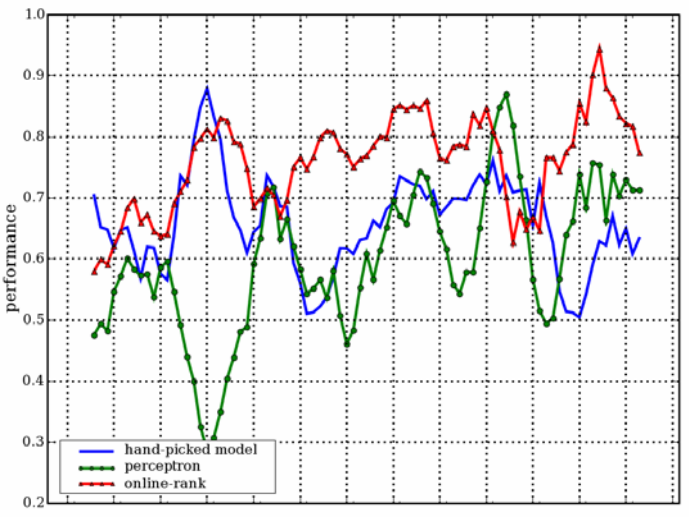
30

Experimental Comparison

- Compare our approach to
  - Using "batch"-trained models
  - Other online learning methods
- Ranking Perceptron
  - Online version
- Hand Picked Model
  - Tuned by humans with domain knowledge

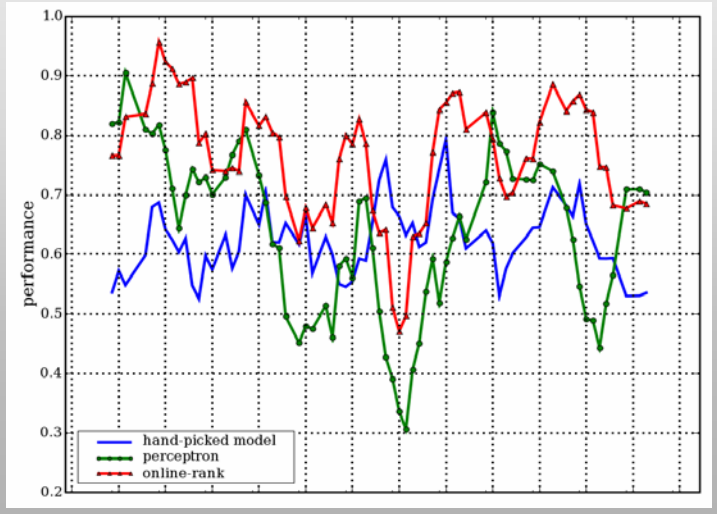
31

Performance – Summer 2005



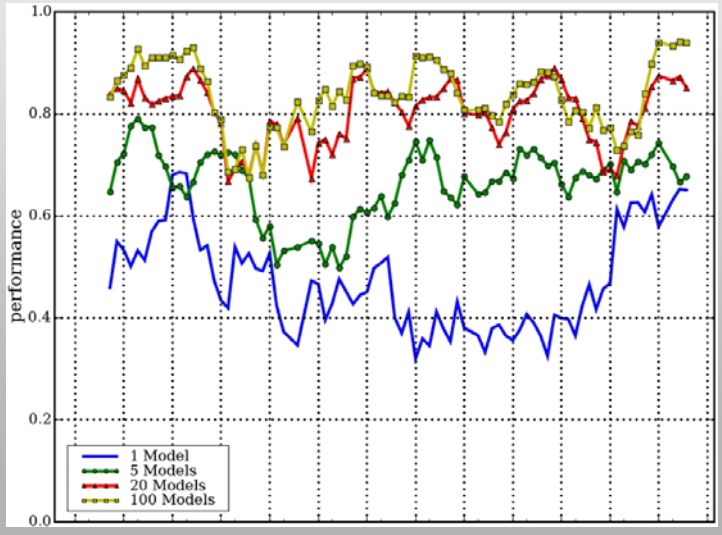
32

Performance – Winter 2006



33

Parameter Variation - Budget



34

Future Work

- Concept drift detection
  - Add new models only when change is detected
- Ensemble diversity control
- Exploit re-occurring contexts

35