1.3

A Random Forest Turbulence Prediction Algorithm

A. Cotter*, J. K. Williams, R. K. Goodrich and J. Craig
National Center for Atmospheric Research, Boulder, Colorado

Abstract

Unlike traditional pilot reports, in-situ EDR reports of atmospheric turbulence from commercial aircraft contain both positive and negative instances, are reported regularly, and have relatively accurate positions and timestamps. These data therefore make it feasible to perform more sophisticated analyses of the causes of atmospheric turbulence than were formerly possible. Several real-time gridded products derived from satellite, radar and numerical weather model data that represent storm location and intensity currently exist. These include quantities such as vertically integrated liquid (VIL), echo tops, and wind direction and velocity. In this paper, the authors present a machine-learning algorithm that predicts in-situ peak EDR based primarily on the values of VIL and echo tops in a spatial neighborhood extending approximately 300km around the measurement point. To summarize the values of the gridded products associated with each in-situ EDR measurement, a set of quantities including distances to grid points with data over certain thresholds, maximum data values within each sub-region and the proportion of grid points over various thresholds within each sub-region were computed. A set of the most useful features for turbulence prediction was then determined using a large-scale automated feature selection algorithm. First, an estimate of the "quality" of each candidate feature was calculated by training a large number of decision trees on small random subsets of candidate features and comparing their performance on a testing set both with and without the feature in question. Then, a linear programming problem was formulated in which the "best" subset of features was chosen under the constraint that no two selected features for a given data source could overlap. The selected features and a large training set were then used to train a random forest as a predictive algorithm. Finally, the performance of the random forest on an independent testing set was evaluated and compared to another turbulence-prediction product.

1. INTRODUCTION

Current turbulence detection/prediction algorithms are primarily *physical* in nature in that they are based on theories of how the causes of turbulence are represented in certain data sets (such as Doppler radar data). Based on these theories, and the relevant data sets, estimates of turbulence may then be calculated. There are many reasons to believe that this is a good approach. One certainly requires some understanding of the atmospheric phenomena that cause turbulence in order to determine, if not “what to look for”, at least “where to look”–that is, which data sources may be expected to be relevant. With this said, it is not necessarily the case that this approach yields the best possible performance. The prediction of turbulence and the understanding of the causes of turbulence are distinct, though complementary, goals. Progress towards the latter goal will, in the long term, likely lead to the progress in accomplishing the former. One need not “understand” turbulence, however, in order to find patterns in the relationship between measured turbulence readings and various “input” data sources (radar data, atmospheric models, satellite data, etc.)–indeed, it is from the inspection of such relationships that theories are created (and validated) in the first place.

If current turbulence prediction algorithms are *physical*, the one described in this paper is *statistical*. Modern machine learning algorithms have become very good at detecting patterns in data and using these patterns to make predictions. Unfortunately, they tend to be “black boxes”–the best-performing algorithms, once trained, may not make the information they have extracted from their training sets

* *Corresponding author address:* Andrew Cotter, Toyota Technological Institute at Chicago, University Press Building, Second Floor, 1427 East 60th Street, Chicago, IL 60637; *e-mail:* cotter@rap.ucar.edu

available in any readily-comprehensible form. This property makes them largely unsuitable to the scientific goal of understanding turbulence, but in no way mitigates their applicability to the purely practical goal of accurately predicting it. In this paper, a machine-learning algorithm for the prediction of atmospheric turbulence is presented, and compared to one currently-implemented algorithm, the NEXRAD Turbulence Detection Algorithm (NTDA).

Section 2 contains descriptions of the data sources used as input to the algorithm described in this paper. In section 3, the particular machine algorithm chosen for this task, random forests, is introduced. Section 4 describes the turbulence prediction algorithm itself, while section 5 examines some results of its training, and its performance on testing data. Finally, section 6 concludes with speculation on how even better performance might be extracted from an algorithm of this type.

2. DATA

2a. *Output Data*

In the past, validation of turbulence prediction or detection algorithms has used pilot reports (pireps) as a source of "truth data". Pireps are difficult to work with, however, because they tend to have extremely poor position and time stamps, and negative turbulence encounters are massively underrepresented in the set of pilot reports relative to their true frequency of occurrence. Cornman, L. B. et al. (1995, 2004) describe the in-situ EDR stream, which is an automated system installed on some United Airlines aircraft that reports, approximately every minute, the median and 90th percentile of eddy dissipation rates (EDRs) encountered in the time since the last report, along with positions and timestamps for every fourth report. The timestamps, unfortunately, have only minutes precision (that is, they contain no seconds portion).

Since the radar data grids–another input to our algorithm–are updated every five minutes, a timestamp error of up to 30 seconds in our "truth data" was undesirable. Hence, the in-situ EDR timestamps were "corrected" to seconds precision using the Aircraft Situation Display to Industry (ASDI) feed. The ASDI feed contains slightly less accurate positions than the in-situ EDR stream, but timestamps that are accurate and recorded with seconds precision. By Kalman-smoothing an ASDI flight track, we first derived a multidimensional Gaussian representing a distribution of what the airplane's position was believed to be at every second. For each of the 60 possible seconds portions of each in-situ point, we then calculated the likelihood of the airplane being at the position reported by the in-situ stream at the given time. We then estimated a distribution on how widely-separated in time two consecutive in-situ points may be (based on our understanding of the reporting algorithm), and used this, in combination with the aforementioned position-likelihoods, as input to a dynamic programming algorithm, which calculated a maximum likelihood sequence of seconds portions of the in-situ EDR points for each flight.

Each in-situ EDR report contains two measures of the turbulence encountered since the previous report–hence, each report applies not to a point in space, but rather to an approximately one-minute flight segment. We used the *midpoints* of these segments as the positions/times of each report, and attempted to predict the aircraft-reported in-situ EDR 90th percentiles (henceforth called "peak EDRs") at these midpoints. The EDR reports themselves are binned into eight classes: 0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65 and 0.75 $m^{2/3}s^{-1}$ (the units will be omitted in the sequel). The 0.05 class contains all points for which the peak EDR reading was less than 0.10, the 0.15 class those between 0.10 and 0.20, and so on up to 0.75, which contains all those greater than 0.70. These classes may be roughly interpreted as follows: 0.05 represent null turbulence, 0.15 and 0.25 are light, 0.35 and 0.45 are moderate, 0.55 and 0.65 are severe, and 0.75 is extreme.

2b. *Input Data*

As has been mentioned, the in-situ peak EDR readings provide the output, or"truth" portion of our dataset. The input portion consists of features derived from the RUC13 model analysis and vertically-integrated liquid (VIL) and echo tops grids provided by Unisys. The RUC13 data consist of a set of three dimensional grids representing a number of atmospheric features derived from a Numerical Weather Prediction (NWP) model. The Unisys VIL and echo tops are two-dimensional grids derived from NEXRAD radar data, each grid point corresponding to a "column" of the atmosphere. VIL represents the total liquid water in the column, in $kg \cdot m^{-2}$, while the echo top is the highest altitude having radar reflectivity of 18 dBZ or greater. In our dataset, only those RUC13 data nearest the aircraft

location at which we wish to predict turbulence are used as features for the prediction algorithm. From the two-dimensional Unisys VIL and echo top grids, however, we calculated a large number of features including not only data *at* the point in question, but also in a region *around* the point.

While the Graphical Turbulence Guidance (GTG, Sharman, R. et al. (2006)) product already utilizes RUC and RUC-derived variables to forecast clear-air turbulence, we expect that the Unisys VIL and echo top grids may improve the prediction of atmospheric turbulence near thunderstorms. The Unisys grids also have considerably higher resolution in both space and time–the Unisys grids are updated approximately every five minutes, in contrast to every hour for the RUC grids. Since one expects convectively-induced turbulence to be a short-lived and possibly highly localized phenomenon, these high resolutions are quite desirable. On the other hand, the RUC data provide information on environmental conditions that may also be significant for turbulence generation by thunderstorms. It is for this reason that that we calculate wide-area features from the Unisys VIL and echo tops while using only the RUC data at the prediction location. The RUC wind direction is also used for the important purpose of orienting the polar grid used to extract radar data proximity and coverage features, as will be explained in section 4a.

3. RANDOM FORESTS

Breiman (2001) introduces random forests, a classification algorithm that is particularly well-suited to the present application. Only the briefest outline of the algorithm will be provided here–the reader is referred to the cited paper for an excellent description of the details.

Random forests make use of an ensemble of weakly-correlated decision trees, which provide a statistical or machine learning technique for learning data classifications. Given a training set T, a decision tree is constructed by first choosing an input feature f and threshold t that split the training set into "maximally distinct" subsets according to some measure (our implementation uses the Gini index (Hastie, T. et al. 2001, p271)). The training set is then split into two subsets $T_<$ and $T_\geq$, the first of which consists of all training samples for which $f < t$, the second for which $f \geq t$. Missing data is handled by treating each missing value of feature f as if it were equal to the median $\hat{f}$ of f over the training set. The decision tree D is defined as a 5-tuple $\left(f, t, \hat{f}, D_<, D_\geq\right)$, where $D_<$ and $D_\geq$ are two decision trees that have been recursively constructed from the data subsets $T_<$ and $T_\geq$, respectively. The recursion stops once no further splits are possible (there may be multiple sample points at such a node, if they have identical values in each input feature). Such leaf nodes are defined as 1-tuples $\left(v\right)$, where the "value" v is defined as the mode of the set of output features in the training set.

For a testing sample, the prediction of a decision tree D will be (again recursively) defined as the prediction of $D_<$ if $f < t$ in the testing sample, and of $D_\geq$ if $f \geq t$. If D is a leaf node, then the prediction is the "value" v of D that was determined during training. As in training, missing data in the testing set is handled by treating a missing f-feature as being equal to $\hat{f}$, the median of f over the *training* set, which was stored in the node D during construction.

A "randomized decision tree", as this phrase will be used in this paper, is extremely similar to a standard decision tree. There are, however, two crucial differences. The first of these is that, rather than being trained on the entirety of the training set, a randomized decision tree is trained only on a random subset. Secondly, when determining which input feature to split on at each node, only a random subset of $\sqrt{n}$ of the n features are considered eligible. This subset is drawn independently for each node, and the best of these eligible features is then selected for splitting.

A random forest is simply a set of some number of randomized decision trees (our implementation used 500). Since each of these trees was trained on a different subset of the training data, and was further randomized by only permitting splits on some random subset of the features at each node, one may hope that the overfitting for which decision trees are justly notorious will "cancel out" over the set of trees–that is, that patterns that are truly present in the training data will be learned by most (or all) of the trees, while those that are simply relics of the sampling will be learned by fewer trees. For a given input, the output of the random forest is a distribution of "votes" over the various data classes. If all goes well, the proportion of trees that "vote" for a given class may be regarded as a measure of the strength of the trend in the training set on which the votes were based. Thus, the random forest general-

izes from patterns occurring in the training set without significant risk of overfitting.

4. TURBULENCE PREDICTION ALGORITHM

4a. *Features*

The goal of the research presented here is to create a machine learning algorithm that uses the RUC13, Unisys VIL and Unisys echo top data to predict the in-situ EDR value at a given point in space p (a latitude/longitude/altitude triplet) and time t. The obvious set of input features would therefore be the value of the grid box containing p taken from the grid active at the time in question, that is, the latest timestamped before t. For the RUC grids, these are exactly the features which we use.

One would expect, however, that the turbulence at p depends on atmospheric phenomena that do not strictly contain p. One might ask, for example: is p "close to" a thunderstorm? If so, is it a large thunderstorm? Is it a severe thunderstorm? Could the edge this thunderstorm have advected over position p in the time since the grid was created? Since we expect answers to such questions to be useful in predicting convectively-induced turbulence, we must provide our learning algorithm with enough data to answer them–hence, we must provide it with features that not only represent the "state of the atmosphere" at p, but also in some region around p.

While one certainly wants to capture as much information as possible in the features chosen, one must attempt to minimize both the number of features, in order for the random forest to operate quickly, and the cost of computing the features. These are competing requirements: we need to maximize the amount of information available in a small number of cheaply-computed features. Moreover, since the ideal outcome of this research would be a real-time system that outputs a 3-D grid of predicted turbulence values or probabilities for use in aviation, computational efficiency is a strict requirement.

The two grids on which we want to compute wide-area features both tend to be locally correlated–that is, the value at each grid point is generally "close to" the values of its neighbors. Keeping in mind that we wish to maximize efficiency in a future real-time system, creating a data structure *once* for each grid is a good investment of computational resources if it allows each wide-area feature on this grid to be calculated more cheaply. After all, in a gridded output product we would need to compute features not just around a single point, but around *every* point in the output grid. Appendix A describes the data structure which was used to efficiently construct three types of features: the closest grid point in some region with value above some threshold; the largest grid value in some region; and the proportion of the points in some region having values in a prescribed interval. These are the three classes of features on VIL and echo tops, called "distance", "upper bound" and "proportion" features, respectively, which are passed as input to the machine learning algorithm described later in this paper.

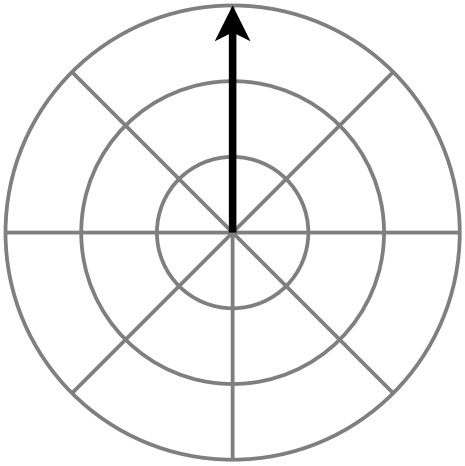

Figure 1: A "dartboard" around a given position. The dark arrow represents the prevailing wind direction, obtained from the RUC analysis, by which the dartboard is oriented.

The selected "regions" and "thresholds" used for the features defined above are now described. For the upper bound and proportion features, the regions were chosen to be ranges of the form $[\theta_l, \theta_u] \times [d_l, d_u]$ (here, $[\cdot, \cdot]$ denotes a closed interval), where the first coordinate is an angle, and the second is a great-circle distance, relative to the point at which we wish to estimate turbulence. For the distance features, the regions were intervals of the form $[\theta_l, \theta_u]$.

The angular coordinate, however, must be specified relative to some reference. For instance, we could take it to be simply the azimuth, so that the nth feature would be oriented in the same compass direction for every point, but we have no reason to be-

lieve that there is anything particularly special about, say, the density of cloud to the North of a point in space, relative to the density of cloud to the East, in terms of their influence on turbulence. It does seem reasonable, however, to expect features upwind of the point to be more important than those downwind. Hence, all angles are measured relative to the wind direction in the RUC model gridpoint nearest the prediction location.

Figure 1 illustrates the sort of regions over which the upper bound and proportion features were computed–a circle of some fixed radius (300km, in our implementation) was first divided into annuli of fixed width, each of which was then further divided into regions spanning some range of angles of fixed size, oriented using the wind direction. The final set of regions resembles a dartboard, and a set of such features will be referred to by this name. Not knowing *a priori* which sizes and positions for the regions would be most appropriate, a set of dartboards were used, and a feature selection methodology was then employed to select the "best" resolution. Each dartboard had radius 300km and was split into a annuli and b angle ranges, for each of $a \in \{1, 2, 4, 8, 16, 32, 64\}$ and $b \in \{1, 2, 4, 8, 16, 32\}$, making a total of $7 \times 6 = 42$ dartboards. The "upper bound" features were defined by taking the upper bound of the Unisys VIL and echo tops in each enclosed region defined by the dartboard. "Proportion" features measure the proportion of grid points in each region having values within a prescribed interval. The Unisys VIL and echo top grids are both bounded (in fact, they are quantized–they take on only a relatively small number of possible values), so, for the proportion features, we generated a set of dartboards for each value range resulting from dividing the set of possible grid values into $c \in \{1, 2, 4, 8, 16\}$ equally-sized ranges, for a total of 31 value intervals, and therefore $31 \times 42 = 1302$ "proportion" dartboards.

The "distance" features were determined similarly. Again oriented by the wind direction, the space around the target point was divided into b equally-sized angular regions for $b \in \{1, 2, 4, 8, 16, 32\}$, as above, and for each of 16 different thresholds (corresponding to thresholds used for the proportion features), the great-circle distance to the closest point within this region and over this threshold was computed.

Finally, four additional features were included, beyond the RUC, VIL and echo top features previously discussed. In the hope of capturing seasonal variations, the month (as an integer in the range $0 - 11$) was included in our feature set. Similarly, so that it would be possible for the random forest to compensate for advection, the age (in seconds) of each of the most recent RUC, VIL and echo top grids were also included. All four of these features turned out to be removed during feature selection, so it appears that they were not as useful as was originally hoped.

4b. *Data Sets*

High turbulence reports are comparatively rare in the in-situ EDR feed. In order to ensure that the training set for our random forest would have enough examples of high turbulence (no machine learning algorithm can learn patterns that are not present in its training set), we sampled our dataset from the EDR stream in such a way that higher peak EDRs were over-represented, relative to their true frequency. During the validation of NTDA, it had been observed that in the vicinity of thunderstorms, the peak EDR readings were distributed roughly geometrically with parameter $\frac{2}{3}$ (that is, the proportion of observations with a peak EDR of 0.05 was approximately $\frac{2}{3}$, of 0.15 was approximately $\frac{2}{9}$, of 0.25 was approximately $\frac{2}{27}$, and so on). In order to choose a higher proportion of high-turbulence readings, yet not change the shape of the distribution on turbulence readings too drastically, we decided to sample our dataset according to a geometric distribution with parameter $\frac{1}{2}$. The dataset was otherwise sampled uniformly randomly, from the set of all in-situ EDR readings between May 10th of 2005 and May 9th of 2006, inclusive. A total of 242125 points were so chosen for training and testing, and the features discussed in section 4a were computed for each.

As a final step, the dataset was divided into several pairs of training and testing sets. In order to ensure that no testing set point was in the same region of space and time as any training set point and thus ensure that the data sets were independent, this division was performed according to the day of the week (which should have no bearing on turbulence). One training-testing set pair, for example, contained as its training set all dataset points that were recorded on a Sunday, Monday or Tuesday, and as its testing set all that were recorded on a Thursday or Friday (all points recorded on a Wednesday or Saturday, the two days separating the training and testing sets, were ignored). Another pair contained as a training set those points from Monday, Tuesday and Wednes-

day, and as a testing set those points from Friday and Saturday. This pattern was repeated (three consecutive days training, one day ignored, two consecutive days testing, one day ignored), until seven pairs of training and testing sets were computed.

4c. *Feature Selection*

Including the distance, upper bound and proportion features derived from the Unisys VIL and echo top grids, the 24 features from the RUC model, and the four additional time features discussed in section 4a, we calculated a total of 514108 features for each point in our dataset. This is obviously excessive, and necessitated a feature-selection step, before any random forest could be trained as a practical predictive algorithm.

Recall that, as described in section 3, our decision tree implementation is capable of handling missing values in both its training and testing sets. Given a decision tree, then, and a testing set, one may estimate the "quality" of a feature f by calculating the performance of the decision tree on the testing set, and comparing this quantity to the performance of the same tree on the same testing set with f set to "missing" (NaN) in every sample.

A straightforward way to perform feature selection would be to initially create a random forest using some random subset of the features, then calculate the value of each feature, remove some number of the worst features, replace them with randomly-selected new features, and iterate. This approach has two main drawbacks. The first is that it would not result in an independent measurement of the value of each feature, making it difficult for us to use the output of feature selection to explore how the atmospheric phenomena recorded in the dataset influence turbulence–we would learn which features were the "best", but *not* the quality of each feature. The second problem is that this approach is somewhat difficult to implement and deploy. Because this algorithm keeps track of a single set of features that evolves over time, to parallelize it (and, due to the size of the task, one *would* need to do so) one would need to synchronize the feature sets between the various participating computers. This is not inherently difficult; however, in the computing environment available for this research, synchronization and automatic distribution of the workload would have been extremely challenging.

Instead, we created a large number (834189) of randomized decision trees, each containing 400 randomly-selected features (each feature was therefore included in approximately 650 decision trees). For each tree, T, we calculated the classification error (the number of misclassifications) $e_{T,c}$, the sum of absolute errors $e_{T,a}$ (where the error for each tree was the difference between the true peak EDR value and the tree's prediction), and the sum of squared errors $e_{T,s}$. We also, for each feature f in each tree T, replaced f with NaN in every testing sample, and calculated errors $e'_{f,T,\alpha}$ for $\alpha \in \{c, a, s\}$, on this new testing set. Once we had done this for all trees, we went through each feature f, and each error metric α, and calculated sums $s_{f,\alpha} = \sum_{T \ni f} e_{T,\alpha}$ and $s'_{f,\alpha} = \sum_{T \ni f} e'_{f,T,\alpha}$ (the subscripts on the sums indicate that they are taken over all trees T which included feature f). Finally, the "with f" sum was divided by the "without f" sum, resulting in numbers $q_{f,\alpha} = \frac{s_{f,\alpha}}{s'_{f,\alpha}}$ representing the factor by which we could expect a decision tree's performance to improve as a result of including feature f. As $q_{f,\alpha}$ is an approximate multiplicative measure of a feature's quality, $\log q_{f,\alpha}$ is an approximate additive measure.

The problem was now to select the "best" subset of n features. This was made more difficult by the fact that many of the features contained redundant information–for example, the minimum distance to VIL over 22.5 over the angle range $[0, \pi]$ is the minimum of the same quantity calculated over each of the two angle ranges $\left[0, \frac{\pi}{2}\right]$ and $\left[\frac{\pi}{2}, \pi\right]$. Including redundant features would be wasteful. Hence, we imposed a constraint that the chosen features be *non-intersecting*–that is, none of the echo top distance features could be taken over intersecting angle ranges, none of the VIL proportion features could be taken over intersecting spatial regions and grid value ranges, and so on. Of course, different *types* of features could intersect freely–echo top distance features could intersect VIL distance features, VIL distance features could intersect VIL upper bound features, etc. These constraints were formulated as a linear programming problem, the objective of which was to maximize the sum of $\log q_{f,\alpha}$ over each chosen feature (we used the classification error qualities $q_{f,c}$), subject to the constraints that at most n features could be chosen, and that they must be non-intersecting. This problem was then solved using the COIN-OR simplex solver (`http://www.coin-or.org/Clp/`).

4d. *Random Forest*

Once the features had been selected (see the results below), a random forest was trained on each training set, and its output (along with the associated "true" peak EDR reading) was recorded for each point in the associated testing set. In our implementation, the forest consisted of 500 randomized decision trees, each of which made a prediction of the turbulence class at each point based on the selected features. The output at each point therefore consisted of the number of decision trees that had "voted" for each class. It is important to note, however, that the distribution of randomized decision tree votes at a particular point does *not* give a true probability distribution on the peak EDR readings at this point.

In a pure classification problem, one would typically take the mode of this "distribution" to be the algorithm's prediction. In the present application, however, we are interested not in the "most likely" turbulence reading (which is overwhelmingly likely to be null), but in the *risk* that the turbulence is above a certain threshold. Taking the distribution of randomized decision tree votes as the output of the algorithm enables us to explore the interplay between the false positive and false negative rates of our predictions using ROC curves, and would further permit us to make "pure classification" predictions by balancing some cost, for each turbulence reading, of a false positive or a false negative. In an operational algorithm, these ROC curves would be used to select a threshold (or thresholds) appropriate to the application.

5. RESULTS

5a. *Feature Selection*

Due to a software bug, the "quality" ratios defined above were not calculated for the distance features. Fortunately, there were few enough such features (2016) that it was relatively straightforward to manually select a subset of 144 that could be expected to well summarize the data contained in the complete set. With this done, 481 RUC, upper bound and proportion features were selected using the approach outlined in section 4c, using the quality estimates derived from classification error. Combining these led to a final set of 625 features. As was previously mentioned, none of the four "timing" features (the current month, and the grid ages) were selected. The selected RUC features are listed in table 1.

Selected	Not selected
Altitude	Wind Azimuth
Wind Velocity	CLWMR
VPTMP	SNMR
MIXR	TKE
VVEL	TSOIL
RWMR	PRATE
ICMR	NCPCP
GRMR	ACPCP
NCIP	PWAT
MSLMA	POT
SHTFL	CAPE
	CIN

Table 1: Selected/not selected RUC features. The altitude, wind azimuth and wind velocity are not "true" RUC features–the altitude field was derived from the pressure altitude contained in the in-situ packet, corrected to a mean-sea-level (MSL) altitude using the RUC data. The wind azimuth and velocity were derived from the UGRD and VGRD RUC grids.

It was decided that the classification error was the natural metric to use when optimizing the random forest, since the goal of this research is to produce a classification (i.e., turbulence probability) algorithm, not a regression algorithm. However, squared errors are more natural for human inspection of feature quality. Figure 2 contains a plot of the estimated quality of the features in one particular dartboard (one of 2688 such dartboards for the various resolutions and thresholds), using the squared error metric. The most notable property of this plot is that, for the vast majority of features, randomized decision tree performance was hurt, rather than helped, by their inclusion. Such features will henceforth be called "bad", and, by extension, features the inclusion of which tended to improve decision tree performance will be called "good". The dominance of bad features is by no means unique to this plot: in fact, only about 14% of all features were good, as measured using the squared error metric. This is likely due to the notorious tendency of decision trees to overfit their training sets. We can also see that the best features tended to be upwind of the point at which turbulence was to be predicted, which is again true in general (approximately 62% of all good features were upwind). This validates our decision

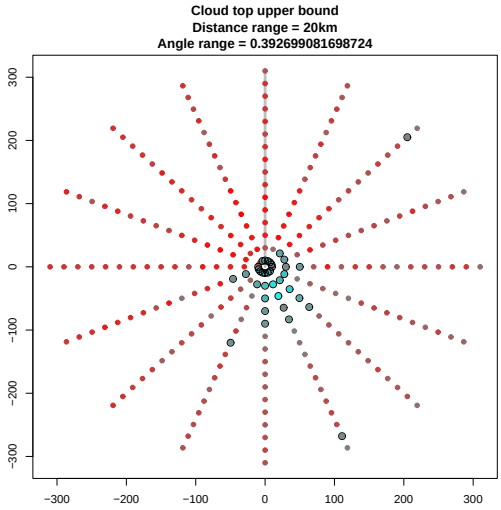

Figure 2: Plot of "cloud top upper bound" feature qualities. The wind direction is the gray line pointing upwards. Each point represents a single feature computed over a region centered on the point. Redder points are worse, and bluer are better. Circled points are "good", as defined in section 4c.

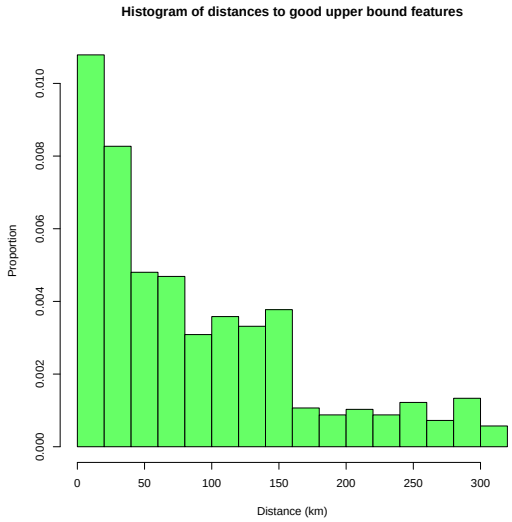

Figure 3: Histogram of distances to "good" upper bound features (both VIL and echo top).

to orient the dartboards by the RUC model winds.

It was also observed that for the upper bound features, as illustrated in figure 3, the vast majority of good features were close to the target point (63% within 100km, 88% within 200km). For proportion features, as shown by figure 4, this was not the case–the good features were roughly evenly distributed over distance ranges. This is possibly due to the fact that nearby features cover a smaller area than more distant ones, so that for proportion features, which would seem to be most appropriately used for estimating the *size* of a storm, the diminishing importance of the more distant features is offset by their increasing extent. For upper bound features, which one would expect to be most useful for estimating a storm's *intensity*, there is no such trade off.

Finally, the reader may have noted that in figure 2, while the best features are upwind and nearby and distant features in all directions are largely irrelevant (note their gray tint in the plot), the worst features seem to be concentrated downwind, and close to the target point. This may be due to the aforementioned tendency of decision trees to overfit–one would expect nearby features in all directions to be correlated with turbulence, if only because if an air-

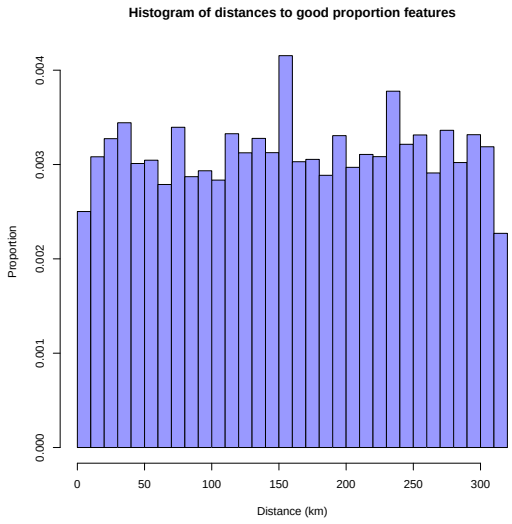

Figure 4: Histogram of distances to "good" proportion features (both VIL and echo top).

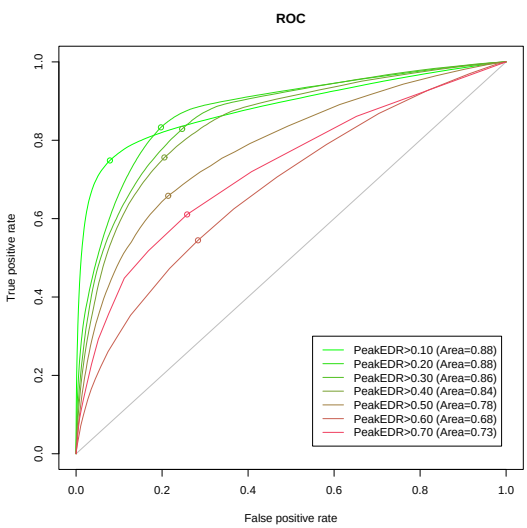

Figure 5: ROC curves of the performance of the random forest algorithm in differentiating points with in-situ peak EDRs less than certain thresholds from those with peak EDRs above, where the testing set has resampled EDR classes distributed approximately geometrically with parameter $\frac{1}{2}$.

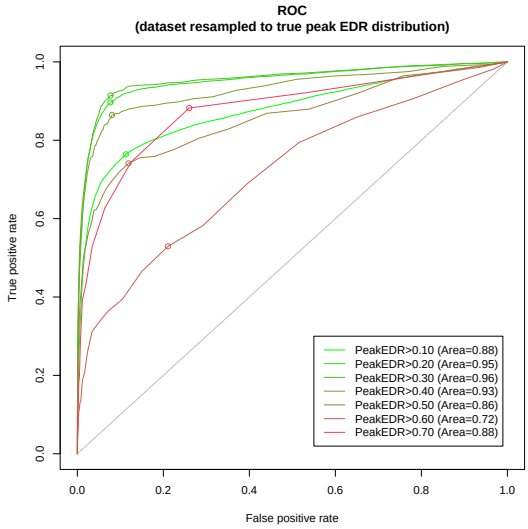

Figure 6: ROC curves of the performance of the random forest algorithm in differentiating points with in-situ peak EDRs less than certain thresholds from those with peak EDRs above, where the testing set has EDR classes distributed approximately according to their true distribution in the in-situ stream.

plane is inside a thunderstorm, it will by definition be surrounded by cloud. Hence, a decision tree will naturally use these features to discriminate between different turbulence intensities. However, these features may have little *predictive* value, since as one might expect, and as has been suggested by the feature selection results, it is primarily the upwind features that have a causal relationship with turbulence. This might explain the low quality scores of these downwind features–perhaps they were considered important by the decision tree during training, but were found during testing to have led to erroneous turbulence predictions.

5b. *Prediction Algorithm Performance*

Once we had selected the 625 most useful features, seven random forests of 500 randomized decision trees were trained, one on each of our seven training sets. Each testing set was then run through the random forest that had been trained on the corresponding training set. From the resulting forest output at each testing point, one can easily calculate the proportion of trees that predicted an EDR reading above some threshold e–this will be represented as $Q\left(E > e\right)$. For each e, one may then produce a ROC curve by varying another threshold t from 0 to 1, counting a positive detection of turbulence of intensity greater than e whenever $Q\left(E > e\right) > t$ and a negative detection when $Q\left(E > e\right) < t$. Figure 5 contains ROC curves of the algorithm's performance computed in this manner.

Recall that our dataset had been resampled in such a way that high peak EDR readings were overrepresented, relative to their true proportion in the in-situ turbulence reports. In order to see how well this algorithm would perform in a "real-world" setting, the testing sets were resampled to contain true peak EDR readings distributed according to the original distribution of the in-situ peak EDR reports. Figure 6 contains the resulting ROC curves. The marked improvement in these results indicated by the higher true positive rates and lower false positive rates may initially be surprising, since one might expect that a ROC curve should be approximately invariant under resampling of the two classes that it is comparing. It is important to note, however, that by resampling our dataset, we are not only changing the total number of examples, but are also changing the proportions among categories, which directly affects the contingency tables on which the ROC plots are based.

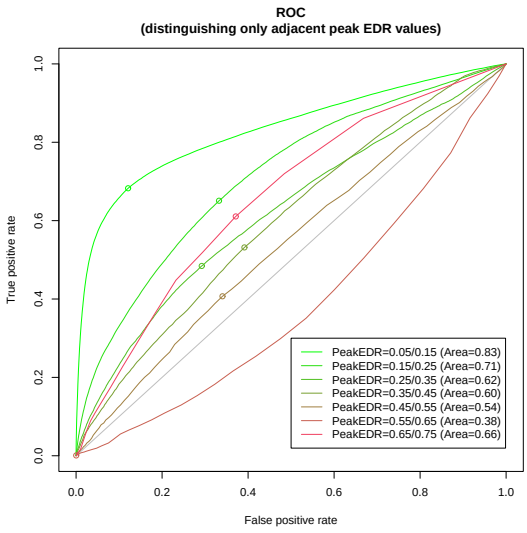

Figure 7: ROC curves of the performance of the random forest algorithm in differentiating between points with in-situ peak EDRs in each of two adjacent classes.

The observation that the shape of ROC curves may change drastically in response to changes in the distribution of the testing set suggests that it might be worthwhile to create a series of ROC curves that *will* be invariant under a resampling of the testing data based on peak EDR readings. Figure 7 contains ROC curves of the performance of the random forest where the task is not to differentiate between those points containing peak EDR readings below some threshold, and those above–rather, the task here is to differentiate between the two classes of points determined by two *adjacent* EDR readings. These ROC curves, it may be hoped, will be easy to compare with those produced by any future algorithms that validate their performance using in-situ EDR readings, regardless of how the testing set is sampled.

One would expect differentiation of adjacent peak EDR readings to be significantly more difficult than merely differentiating "high" from "low", and indeed the ROC curves in figure 7 bear out this expectation, though most of them show reasonably good skill, for such a difficult task, particularly in the important distinction between light and moderate turbulence.

The fact that the ROC curve for differentiation between peak EDR readings of 0.55 and 0.65 is negative is surprising, and may be due in part to the small number of high-turbulence training examples and the small relative difference between these turbulence levels. The comparatively high performance of the algorithm in differentiating between points with peak EDRs of 0.65 from those of 0.75 is easier to explain. In the in-situ feed, the 0.75 class contains all points with peak EDR readings of 0.70 *or higher*. Thus, it contains a number of extremely high, and hence distinctive, turbulence cases.

Threshold	NTDA	RF (5)	RF (6)
0.10	0.64	0.88	0.88
0.30	0.74	0.86	0.96
0.50	0.78	0.78	0.86
0.70	0.86	0.73	0.88

Table 2: Areas under the ROC curves for discriminating in-situ peak EDR readings below certain thresholds from those above. The "RF (5)" and "RF (6)" columns contain the areas under the ROC curves in figures 5 and 6, respectively.

5c. *Comparison with NTDA*

Figure 12 in Williams, J. K. et al. (2005) contains ROC curves measuring the performance of an experimental version of the NTDA in discriminating between peak EDR readings above and below certain thresholds (0.10, 0.30, 0.50 and 0.70,representing light, moderate, severe and extreme thresholds). The testing set used in the NTDA validation was sampled from in-situ EDR reports without regard to their peak EDR values, *except* that out-of-cloud cases (where there was no NTDA data available) were ignored. As was previously noted, it was observed that, in the vicinity of storms, peak EDRs appear to be distributed approximately geometrically, with parameter $\frac{2}{3}$. Since figure 5 was created using a testing set for which peak EDR readings were distributed geometrically with parameter $\frac{1}{2}$, while figure 6 was created using a testing set where the distribution of peak EDR readings was approximately equal to their true frequency in the in-situ EDR stream, one would expect the distribution of peak EDRs in the NTDA validation set to lie somewhere between these two extremes. Table 2 contains a comparison of the areas under the ROC curves from the NTDA validation paper, and those in figures 5 and 6. Although ROC curves from different datasets may not be directly comparable, the random forest developed in this paper appears to outperform the NTDA in the prediction of light (a threshold of 0.10) or moderate

(0.30) turbulence, was slightly better than NTDA at the prediction of severe (0.50) turbulence, and was slightly worse at the prediction of extreme (0.70) turbulence. These results are striking when one considers that the NTDA is a direct detection algorithm. Of course, a more meaningful comparison between the two algorithms would require testing their performance on precisely the same datasets. but this comparison has not yet been performed.

6. SUMMARY AND FUTURE WORK

A machine-learning technique has been presented for effectively predicting atmospheric turbulence based on RUC model data and the two-dimensional Unisys grids of VIL and echo tops. Evaluation results show that it compares favorably to an existing turbulence detection algorithm, and the ROC performance curves suggest that it could be a valuable operational product. It has long been understood in the AI community that perhaps the key component to the successful application of a machine learning algorithm to a particular problem is the choice of "the right set of features", and it appears that, in this particular case, the wide-area features computed from the Unisys grids, as described in section 4a, are appropriate to the task of turbulence prediction. With this said, the feature selection algorithm of section 4c leaves much to be desired. The most obvious route to improving the random forest turbulence prediction algorithm would be to implement the "straightforward" feature-selection technique described in section 4c.

Improvement might also be achieved by providing the random forest with a richer set of features, for instance, features derived from NTDA data, satellite data, or raw Doppler radar data. When provided with three-dimensional radar data, such as the radar reflectivity, velocity, or NTDA data, it seems reasonable to expect that the performance of a properly trained random forest could only improve. In addition, a random forest based on radar data might be used to improve the performance of the NTDA itself, thus offering the prospect of providing better in-cloud turbulence measurements to the random forest turbulence prediction algorithm.

Unfortunately, there are significant challenges to the inclusion of three-dimensional data, the greatest of which is simply its *size*–for instance, the raw radar data to cover our current 1-year, CONUS dataset would occupy several terabytes. Using 3-D mosaic grids of the radar data would make storing it more tractable, and such grids would also be necessary to collect the dartboard data using an algorithm like the one described in section 4a. Since the Doppler radar dartboards would be three-dimensional, the number of candidate features would potentially be much higher than the corresponding number for the 2-D Unisys grids, making an alternative to the linear-programming feature selection method presented here particularly desirable.

To use the random forest operationally would require implementing it in a real-time system. Efficiency and parallelism have been key considerations at every step of its development, and it seems possible that, with sufficient computational resources, it could be used to generated high-dimensional turbulence grids every five minutes over the continental United States. These grids, like those of GTG or NTDA, could then be used to provide turbulence information to airline dispatchers, pilots, and air traffic managers. Due to the success that has been shown at this early stage, it seems reasonable to expect that such a system would show impressive performance.

Acknowledgement Thanks to Valliappa Lakshmanan of NSSL for making his AMS Latex style file available on the web.

Acknowledgement This research is in response to requirements and funding by the Federal Aviation Administration (FAA). The views expressed are those of the authors and do not necessarily represent the official policy or position of the FAA.

References

Breiman, L., 2001: Random forests. *Machine Learning*, **45**, 5–32.

Cornman, L. B., Meymaris, G., and Limber, M., 2004: An update on the FAA aviation weather research program's in situ turbulence measurement and reporting system. *11th AMS Conference on Aviation, Range, and Aerospace Meteorology*.

Cornman, L. B., Morse, C. S., and Cunning, G., 1995: Real-time estimation of atmospheric turbu-

lence severity from in-situ aircraft measurements. *Journal of Aircraft*, **32**, 171–177.

Hastie, T., Tibshirani, R., and Friedman, J., 2001: *The Elements of Statistical Learning*. Springer-Verlag, New York, NY.

Sharman, R., Tebaldi, C., Wiener, G., and Wolff, G., 2006: An integrated approach to mid- and upper-level turbulence forecasting. *Weather and Forecasting*, **21**, 268–287.

Williams, J. K., Cornman, L. B., Yee, J., Carson, S. G., and Cotter, A., 2005: Real-time remote detection of convectively-induced turbulence. *32nd AMS conference on radar meteorology*.

A. CALCULATION OF WIDE-AREA FEATURES

The data structure which was used to accelerate the calculation of wide-area features is related to a quadtree. For an $n \times m$ input grid G, where the element in the ith row and jth column will be represented as $(G)_{i,j}$, we may define two sequences L_k and U_k of size $n_k \times m_k$, each element of which is a lower (respectively upper) bound on a sub-grid of G. First, let $L_0 = U_0 = G$, $n_0 = n$ and $m_0 = m$. Then, iteratively define for $k \geq 0$:

$$\begin{aligned} n_{k+1} &= \left\lceil \frac{1}{2} n_k \right\rceil \\ m_{k+1} &= \left\lceil \frac{1}{2} m_k \right\rceil \\ (L_{k+1})_{i,j} &= \min_{\substack{i' \in \{2i, 2i+1\} \\ j' \in \{2j, 2j+1\}}} (L_k)_{i',j'} \\ (U_{k+1})_{i,j} &= \max_{\substack{i' \in \{2i, 2i+1\} \\ j' \in \{2j, 2j+1\}}} (U_k)_{i',j'} \end{aligned}$$

with the understanding that elements of the $\min/\max$ that are outside the bounds of L_k and U_k are ignored. Once L_k and U_k are 1×1 grids, we stop. We define K as the smallest value of k for which $n_k = m_k = 1$.

This data structure allows us to compute a number of features of interest quickly. The general method of doing so is to search for the set of all grid points satisfying some property by first finding some candidate regions using the bounds contained in the coarsest grids, then progressively refining our search on ever-finer bound grids. For example, for some point p and region of space S (represented as a set of pairs of G-indices), we may find the distance from p to the closest point in S with a value larger than some threshold, α, as follows. First, define:

$$H_{k,i,j} = \left\{ (i', j') \in \mathbb{N}^2 : \begin{array}{c} 2^k i \leq i' < 2^k (i+1) \\ 2^k j \leq j' < 2^k (j+1) \end{array} \right\}$$

This will be exactly the set of indices of points in G that were used to calculate $(L_k)_{i,j}$ and $(U_k)_{i,j}$ (again, with the understanding that points outside the bounds of G are ignored). Now let $d\left(p, i, j\right)$ be a metric giving the distance from a point p to a grid point $\left(G\right)_{i,j}$, and suppose (as is the case for the great-circle distances used in our implementation) that we may quickly find lower and upper bounds on the distance to the points in $H_{k,i,j}$:

$$\begin{aligned} l_k\left(p, i, j\right) &\leq \min_{\left(i', j'\right) \in H_{k,i,j}} d\left(p, i', j'\right) \\ u_k\left(p, i, j\right) &\geq \max_{\left(i', j'\right) \in H_{k,i,j}} d\left(p, i', j'\right) \end{aligned}$$

We will define a sequence of lists $A_K, A_{K-1}, \ldots, A_0$, each of which contains the region of the grid in which we are "interested" at each iteration. Let $A_K = \{(0,0)\}$, representing that initially we will search over all grid points in $H_{K,0,0}$–namely, the entire grid. We may now define an iteration for $k : 0 \leq k < K$. First:

$$b_k = \min \left\{ u_{k+1}\left(i, j\right) : \begin{array}{c} (i,j) \in A_{k+1} \\ (L_{k+1})_{i,j} \geq \alpha \\ H_{k+1,i,j} \subseteq S \end{array} \right\}$$

This will be an upper bound on the distance to some point that we *know* is inside S, and is no smaller than α. If the set over which the minimum is taken is empty, let $b_k = \infty$. We may now define:

$$A_k = \left\{ (i,j) : \begin{array}{c} \left(\left\lfloor \frac{1}{2} i \right\rfloor, \left\lfloor \frac{1}{2} j \right\rfloor\right) \in A_{k+1} \\ (U_k)_{i,j} \geq \alpha \\ l_k\left(p, i, j\right) \leq b_k \\ S \cap H_{k,i,j} \neq \emptyset \end{array} \right\}$$

This will be exactly the set of regions of interest in the "next" set of bounds that could possibly contain points larger than α inside S that are closer than b_k.

This completes the definition of a single iteration. We perform these iterations until we have computed A_0, which contains a list of grid points with thresholds larger than α (since $U_0 = G$) that are inside S (since $H_{k,i,j} = \{(i,j)\}$); if this set is nonempty, one of its members is the closest point. We now simply compute the distance to each point, and choose the one nearest to p.

In practice, due to the smoothness of the input grids in question G (Unisys VIL and echo tops), the lower and upper bounds are frequently close together (especially for the finer grids, though to some extent in all of them), which enables us to rule out large regions of the grid in early iterations using both to the distance bound b_k (which is defined using the grid-value lower bound L_{k+1}), and the grid-value upper bound U_k (in the definition of A_k for $0 \leq k < K$). Similar (though slightly more complicated) algorithms were developed to use this data structure to quickly compute the largest grid value in some region and the proportion of the points in some region having values in a prescribed interval.