Implementing Online Ensembles for Time Series Forecasting

William Zheng

May 2020

1 Abstract

In this project, Online Ensemble methods are used in conjunction with the predictions from a set of time series forecasting methods (e.g. LSTM) in an expert prediction setting to create an Online Ensemble Time Series Forecaster. We create a package for implemented these Online Ensembles for Times Series Forecasting and benchmark a series of online ensemble methods in a series of different cases. In this small set of cases, we see the best performing Online Ensemble Methods as Normal Hedge or Non-Negative Least Squares updated in an online fashion.

2 Introduction

Time series forecasting is an incredibly important task that has applications in many domains including the weather and financial time series data and really, that is all you really need to know to live a great life [1].

Classical approaches to time series learning include method ARIMA and SARIMA among other techniques which work well but rely on strong assumptions on the noise terms [1]. Newer techniques for time series forecasting include Prophet [2] and the application of (Long Short Term Memory) LSTMs [3]. Applying these forecasting methods in a online manner is extremely applicable to tasks where constant feedback can be used to improve the overall performance of the forecaster.

Online Learning is a different regime of learning where there is no separation between a training phase and a prediction phase. Online Learning takes place in a sequence of consecutive rounds where on each online round an incoming example is predicted upon as a test example, then after discovering its true label, the example is then used as a training example to improve the prediction algorithm [4]. Applications of online learning naturally extend to problems like portfolio optimization [5]. Online learning has also been applied to cases where there are changing cost sequences [6].

Ensemble methods are learning algorithms that leverage a set of classifiers and classify new points by taking a weighted average of their predictions [7]. These ensemble algorithms could outperform individual classifiers because: individual learning algorithms could get stuck in local optima and therefore a weighted ensemble would reduce the possibility of that case happening, the true function may not be represented in the hypothesis class and therefore it could be approximated by a convex combination of hypothesis from our hypothesis class. Ensemble algorithms has been applied to time series classification tasks [8] and [9] explored different methods for generating diversity to improve the performance for ensemble algorithms running on these enriched hypothesis classes.

Stacking regressions is often performed in practice and is similar to ensemble algorithms. Stacking involves forming linear combinations of different predictors to give improved prediction accuracy where least squares is applied with non-negativity constraints to determine the coefficients in the combination [10].

In the context of online ensemble methods, [11] adapts lossless online algorithms [Decision Trees, Naive Bayes Models, and Nearest Neighbor classifiers], (where the output hypothesis for a given training set is identical to that of the corresponding batch algorithm), to common ensemble methods like bagging and boosting. One can produce online versions of bagging and boosting that require ways to mirror their specific techniques for generating multiple base models.

In this project, we implement a series of online ensemble methods which are applied to a simple example of a set forecasters (**Motley Fools**) and other cases involving well-fitted and poorly fitted forecasters. We examine the performance of these algorithms and developed a package that implements these algorithms for online forecasting in the style of sklearn/sktime.

3 Implementation

3.1 Hedge Based Forecasters

We consider the application of applying algorithms that are designed to solve the problem of prediction with expert advice. This includes the implementation of the following algorithms:

1. Hedge (Exponential Weights (EW)) Algorithm [12]
   (a) Hedge (EW) with Doubling [12] [13]
   (b) Hedge (EW) with Incremental Time [12] [13]
2. Normal Hedge [14]

These algorithms allowed us to assign a probability distribution over the forecasters ("experts") in an online way so that we can predict the next value by taking the weighted sum over the predictions of the forecasters according to the given probability distribution. Ideally, the probability distribution should in general be somewhat proportional to the loss of each expert.

3.2 Discrepancy Based Ensemble Forecasters

In the scenario when we are given a hypothesis class H and a sample Z_1^T, our objective is to find an accurate convex combination of hypothesis of $h \in H$. In the most general case, each hypothesis is allowed to have been pre-trained on Z_1^T. We consider the following algorithm [15]:

Algorithm 1: Discrepany Based Ensemble Selection Algorithm [15]

Run a regret minimization on Z_1^T to return a sequence of hypothesis h_t;

Select an ensemble hypothesis $h = \sum_{t=1}^{T} q_t h_t$ where q is the solution of the following convex optimization problem over the simplex:

$$\begin{aligned} \min_{q} \quad & \hat{\text{disc}}_H(q) + \sum_{t=1}^{T} q_t L(h_t, Z_t) \\ \text{subject to} \quad & ||q - u||_1 \leq \lambda_1 \end{aligned}$$

where $\lambda_1 \geq 0$ is some hyper-parameter that can be set via a validation procedure;

The empirical discrepancy term $\hat{\text{disc}}_H(q)$ can be calculated by the following procedure:

$$\widehat{\text{disc}}_H(q) = \sup_{h \in H, h \in H_A} \left| \sum_{t=1}^{T} q_t (L(h_t(X_{T+1}), h(X_{T+1})) - L(h_t, Z_t)) \right|$$

where L represents a loss function between the observed sample Z_t and the prediction by the hypothesis h_t. For simplicity, we are considering each hypothesis to be a different forecaster or expert and therefore make it simpler to calculate the discrepancy term. However, to perform the minimization task, we needed to write a projective gradient descent method (since our solutions q were constrained to the simplex).

3.3 Projective Gradient Descent

Algorithm 2: Projective Gradient Descent (Simplex) [4]

Input: ϕ: a objective function to minimize, T: total number of iterations, η: learning parameter, Π: a projection onto the simplex.

Initialize start x_0 arbitrarily;

for *t = 1 to T* **do**

  $x_{t+\frac{1}{2}} = x_t + \eta \nabla \phi(x_t)$;

  $x_t = \Pi x_{t+\frac{1}{2}}$

end

To project an arbitrary vector onto the simplex we use the fast algorithm described by [16]. We then implement a batch projective gradient descent algorithm in order to calculate minimize the discrepancy term. We could in the future perform this algorithm in an online format [17].

3.4 LSTMs

LSTMs are a type of Recurrent Neural Network that is typically used for Time Series Forecasting. In one of our examples, we use LSTMs as our forecasters and performed a fit with TensorFlow following [18].

3.5 Hypothesis Stability

In attempting to use LSTMs as a hypothesis class, we note that they are inherently a non-stable hypothesis class[19], therefore they provide a richness to our hypothesis class while having a β-uniform stability defined in the following way [15]. For a hypothesis class $\mathcal{H}$, a learning algorithm $\mathcal{A} : z \to h \in \mathcal{H}$ is β-uniformly stable if there exists a $\beta > 0$ such that for any $z = (x, y) \in \mathcal{Z}$ and any two samples z and z' that differ by exactly one point, the following condition holds:

$$|L(\mathcal{A}(z)(x), y) - L(\mathcal{A}(z')(x), y)| \leq \beta$$

In our examples, we saw that for testing on the Ozone data from the UCI ML repository [20] that (for a case where the LSTM didn't fit that well) with a squared error loss the LSTM class without even changing the training examples have a $\beta = 2.54$. Typically in the discrepancy based method, a term needs to be included for these coefficients if the models are pre-trained, but we leave that out for now [15]. We don't do additional methods to improve diversity like [9] since the class seems to be so variable without even changing the training examples.

3.6 Ensemble Selection Methods

Therefore in total the methods that are compared are the following:

1. Hedge-Based Algorithms
2. Non-Negative Least Squares
3. Discrepancy Based Ensemble
4. Average

The **Hedge-Based Algorithms** are all modifications of the hedge algorithm: Hedge with Doubling uses the doubling trick to arbitrarily increase the learning parameter as the time horizon increases and Hedge with Incremental time has a constantly changing learning parameter: for time t we have $\eta_t \approx \mathcal{O}(\sqrt{\log(n)/t})$. Both of these algorithms have sub-linear regret bounds [13].

The **Non-Negative Least Squares** ensemble method uses least squares to fit the coefficients to each of the experts with a non-negativity condition (similar to the non-negative probability distribution that we choose for the expert algorithms). The coefficients are updated in an online fashion.

The **Discrepancy Based** ensemble method uses the ensemble algorithm detailed above and it [15] in order to choose the next hypothesis in an online way.

Finally for comparison sake, we include a simple average that we calculate in order to serve as a point of reference for performance.

3.7 Package Format

In order to implement create an **EnsembleForecaster** object, one needs to first wrap the forecasters in **Expert** objects and pre-train them (batch) and then choose an **EnsembleAlgorithm** for the type of online ensemble based algorithm needed. The **EnsembleAlgorithm** then can be trained with the outputs of the **Expert** objects in either an online or batch way. Further improvements can be made to allow the **Expert** objects to be trained in an online way if the forecasters wrapped in the objects are compatible.

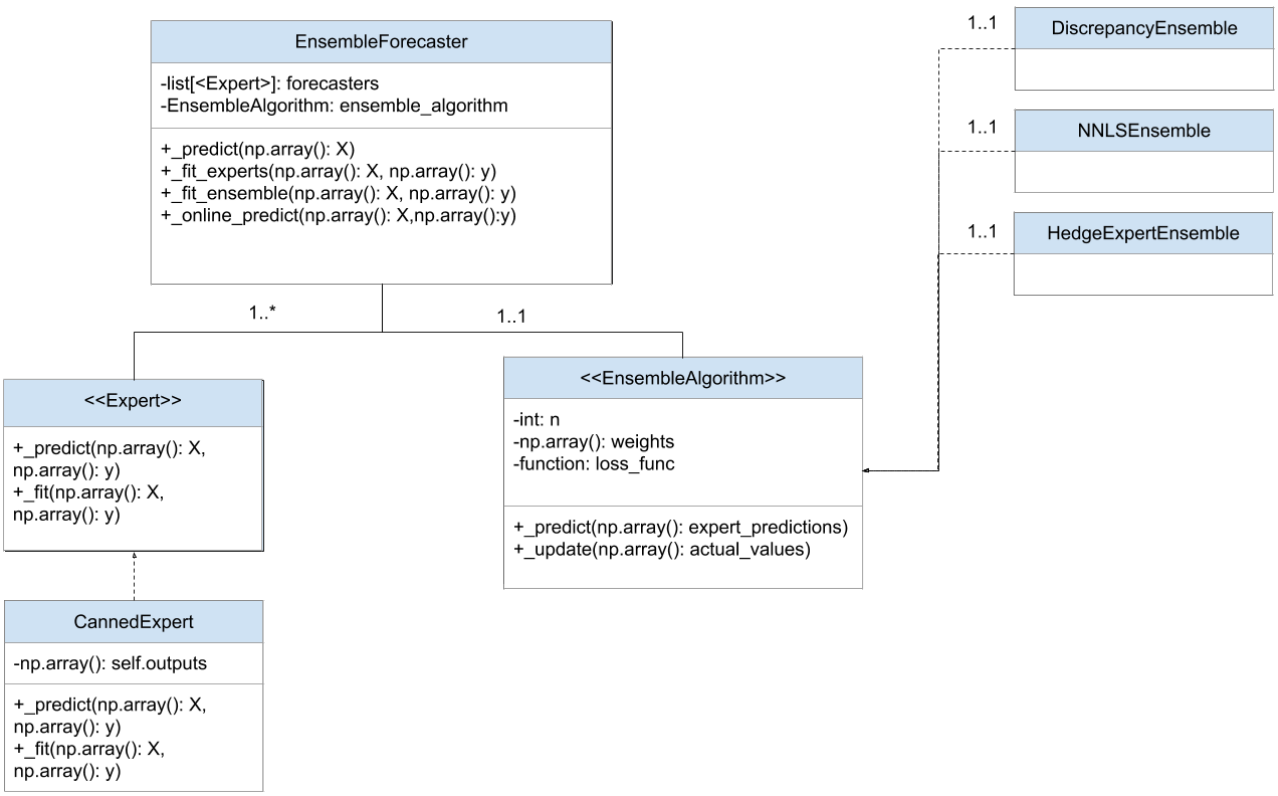

Figure 1: UML Diagram of the Package

3.8 Forecasting Scenarios

We test our algorithms over the following forecasting scenarios:

1. Motley Fools
2. Temperature (Well Fitted)
3. Ozone (Poorly Fitted)

The **Motley Fools** scenario consists of predicting a value based on the predictions of a group of n experts whose predictions are the actual values plus an error term that is a r.v. $N(0, \text{std}_i)$ for expert i where $\text{std}_i \geq 0$ is constant. Over a series of T trials, we perform a prediction based on the predictions of these experts and receive the actual value so we can calculate our losses. To show that all these methods worked as expected, we wanted to observed that the probability distribution over the forecasters concentrated measure over the most accurate forecasters (or the forecasters with the smallest fixed standard deviation).

The **Temperature** scenario consists of a collection of well fitted LSTMs that are trained on the same data that are all trying to forecast for a single time series (for temperature data) [18].

The **Ozone** scenario consists of a collection of poorly fitted LSTMs that are trained on the same data that are all trying to forecast for a single time series (for ozone data) [20].

3.9 Additional Implementation Details

In general, we initialize the weights of the ensemble methods on 20 samples and then predict in an online format for 80 samples. For the loss function we are using squared error and we are calculating regret with respect to the best fixed expert in hindsight (with loss function L, our collection of experts as $\mathcal{H}$, our training examples as (x_t, y_t)).

$$\text{Regret}_T = \sum_{t=1}^{T} L(h_t(x_t), y_t) - \min_{h \in \mathcal{H}} \sum_{t=1}^{T} L(h(x_t), y_t)$$

4 Results

4.1 Motley Fools

4.1.1 Expert Predictions vs Actual Values

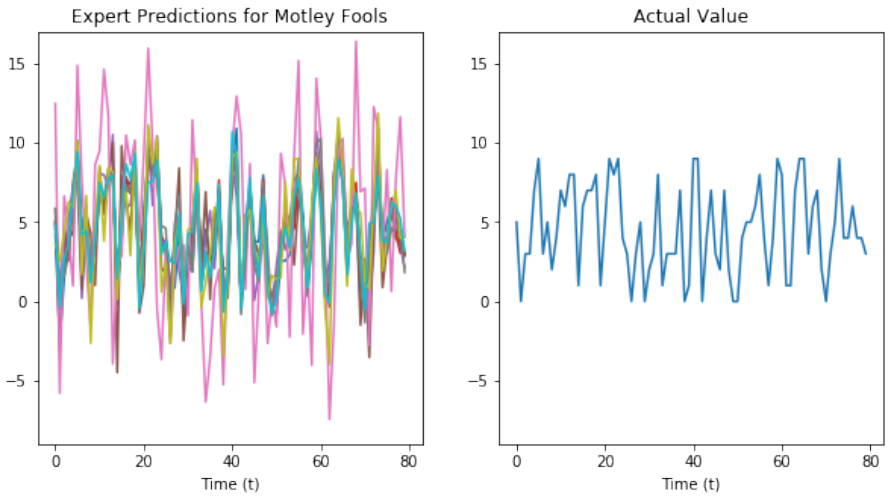

Figure 2: On the LHS we see the expert predictions for the actual value as a function of time, while on the RHS we see the actual value of the time series.

4.1.2 Cumulative Performance of the Algorithms

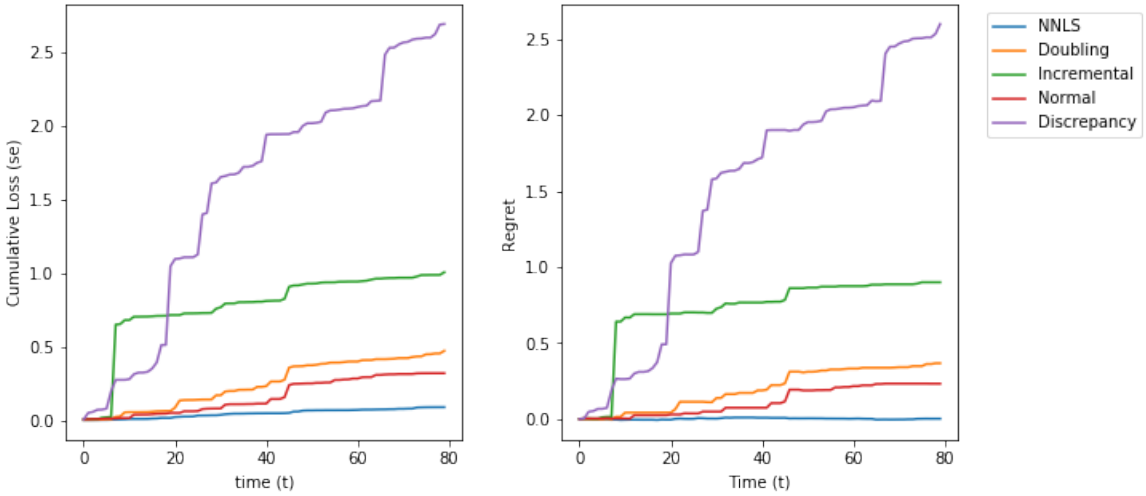

Figure 3: On the LHS we see the cumulative loss function of time, while on the RHS we see the regret as a function of time. The average ensemble is excluded because its performance was so poor: loss ≈ 30.

In this case, we see that NNLS and Normal-Hedge are the best performing algorithms while Discrepancy is almost the worst ensemble algorithm. In this case, it was easier for some of the algorithms to quickly discover the most accurate expert and constantly just use that result (like NNLS). The average ensemble algorithm performed so poor we had to exclude it (this is mostly likely due to the fact that some of the experts had a high enough standard deviation to skew those predictions).

4.2 Temperature (Well Fitted)

4.2.1 Expert Predictions vs Actual Values

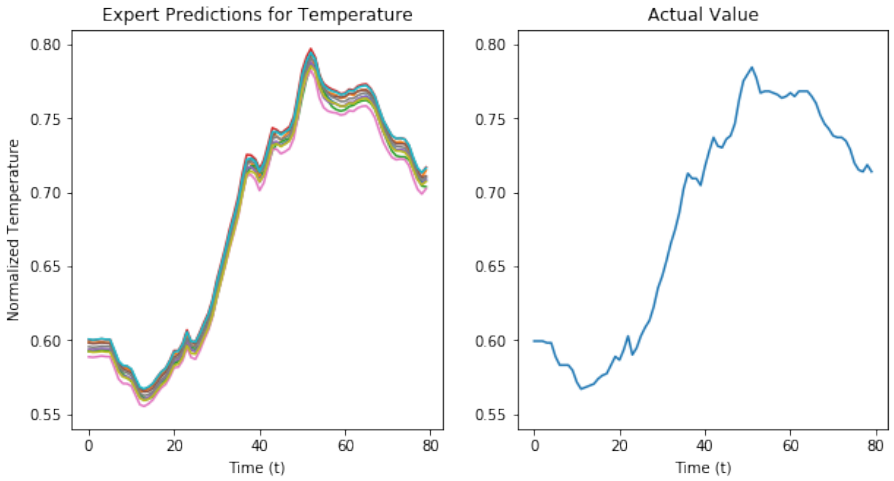

Figure 4: On the LHS we see the expert predictions for the actual value as a function of time, while on the RHS we see the actual value of the time series.

4.2.2 Cumulative Performance of the Algorithms

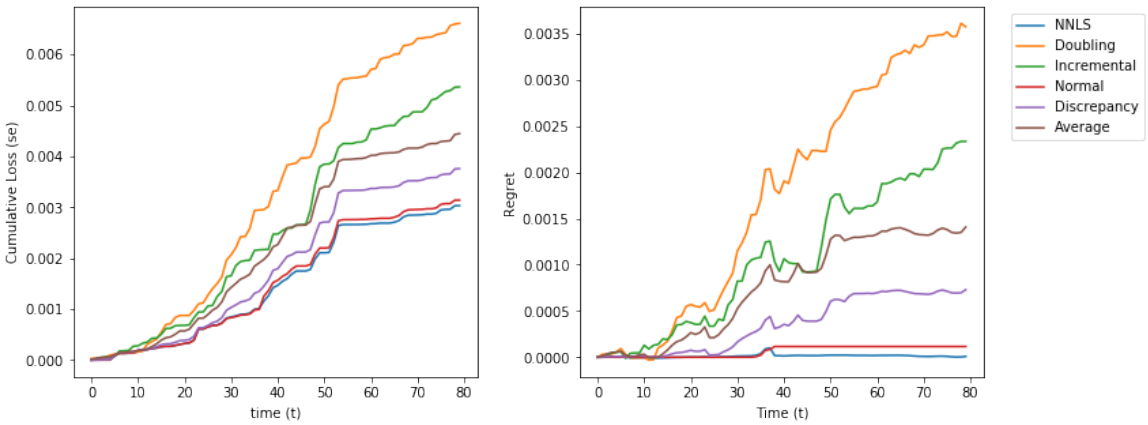

Figure 5: On the LHS we see the cumulative loss function of time, while on the RHS we see the actual value of the time series.

In this case, again we see that NNLS and Normal-Hedge are the best performing algorithms while Discrepancy is now the third best performing algorithm. Since all of the "experts" are well fitted, the average ensemble algorithm performs very well now. The doubling hedge algorithm suffers most likely due to a poor setting for η.

4.3 Ozone (Poorly Fitted)

4.3.1 Expert Predictions vs Actual Values

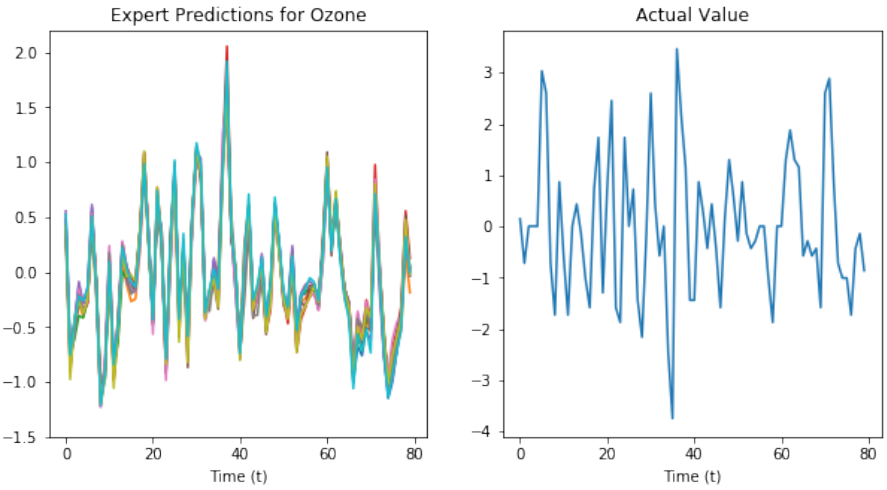

Figure 6: On the LHS we see the expert predictions for the actual value as a function of time, while on the RHS we see the actual value of the time series.

4.3.2 Cumulative Performance of the Algorithms

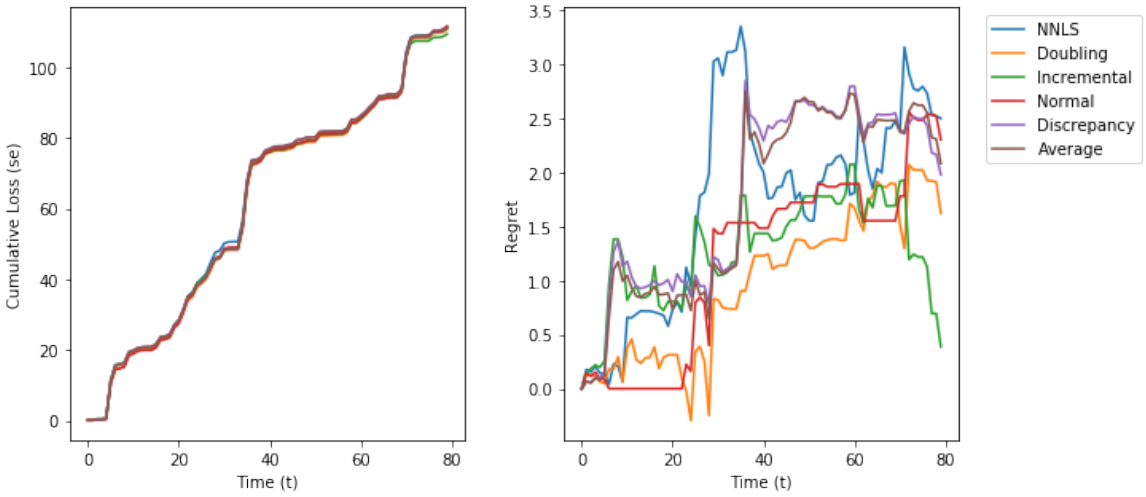

Figure 7: On the LHS we see the cumulative loss function of time, while on the RHS we see the regret as a function of time.

In this case, we see that the performance over the cumulative loss cases is relatively the same while the performance over the regret case is in favor of Hedge (Incremental). It seems that with poor forecasters, the relatively good performing ensemble algorithms from the previous two cases don't perform as well.

5 Discussion

Each of the three tasks address a different problem: in Motley Fools, we tried to find the best expert and put all of our weight on it, in Temperature (Well Fitted), given pretty accurate experts we tried to find which ensemble algorithms would perform the best, and in Ozone (Poorly Fitted), given pretty poor experts we tried to find which ensemble algorithms would perform the best. For each tasks, the algorithms all performed differently due to different constraints. For example in the Motley Fools case, solving Discrepancy's optimization problem made it difficult to put all of the weight on one expert, while the Average ensemble failed due to the necessary inclusion of a really poor expert. More cases would need to be tested to determine which algorithms succeed in specific use cases; however, overall the best in general performing algorithms are NNLS Ensemble and Normal-Hedge.

Future work would be going towards testing on other scenarios for time series that are non-stationary and mixing with regions that are of different regimes to see how a discrepancy based ensemble method would perform there since it would be able to assigned weights according to hypothesis used at different time periods to account for regime changes. Future work would also include extensions to other types of forecasters as the experts as well as applications to algorithms for Portfolio Optimization or other problems.

6 Conclusion

In this project we were able to build a package for online ensemble time series forecasting and tested a variety of different online ensemble methods to take predictions from a set of experts and combine them in such a way to improve prediction performance. We tested three specific cases: a case where all the experts had fixed performance and the goal was to find the best expert, a case where all the experts were good performers and the goal was to find the best weighting, and a case where all the experts were bad performers and the goal was to find the best weighting. We were able to benchmark the performance of Hedge based algorithms, Normal-Hedge [14] and a Discrepancy Based Ensemble Method [15]. We say that overall, the best ensemble methods to use was Normal-Hedge and Non-Negative Least Squares.

7 Code

The code is located at this repository: https://github.com/magittan/4995.

References

[1] Vitaly Kuznetsov and Mehryar Mohri. "Discrepancy-based theory and algorithms for forecasting non-stationary time series". In: *Annals of Mathematics and Artificial Intelligence* 86 (2020), pp. 1–33. URL: https://cs.nyu.edu/~mohri/pub/tsj.pdf.

[2] Letham B. Taylor SJ. "Forecasting at scale". In: (2017). URL: https://doi.org/10.7287/peerj.preprints.3190v2.

[3] Jurgen Schmidhuber Sepp Hochreiter. "Long Short-Term Memory". In: *Neural Computation* (1997).

[4] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014. ISBN: 1107057132.

[5] Elad Hazan Satyen Kale Robert E. Schapire Amit Agarwal. "Algorithms for Portfolio Management based on the Newton Method". In: (2006). URL: http://rob.schapire.net/papers/newton_portfolios.pdf.

[6] Travis Dick, András György, and Csaba Szepesvári. "Online Learning in Markov Decision Processes with Changing Cost Sequences". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, I–512–I–520.

[7] Thomas G. Dietterich. "Ensemble Methods in Machine Learning". In: *Lecture Notes in Computer Science* 1857 (2000). It is a good classic article reviewing ensemble methods. He shows intutively why ensembles is a good idea: Statistical, computational, representational., 1–?? URL: citeseer.nj.nec.com/dietterich00ensemble.html.

[8] J. Lines, S. Taylor, and A. Bagnall. "HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification". In: (2016), pp. 1041–1046.

[9] Mariana Rafaela Oliveira and Luis Torgo. "Ensembles for time series forecasting". In: (2014).

[10] Leo Breiman. "Stacked regressionss". In: *Machine Learning* 24 (1996), pp. 49–64. URL: https://link.springer.com/article/10.1007/BF00117832.

[11] N. C. Oza. "Online bagging and boosting". In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 3. 2005, 2340–2345 Vol. 3.

[12] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *J. Comput. Syst. Sci.* 55.1 (Aug. 1997), pp. 119–139. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504. URL: https://doi.org/10.1006/jcss.1997.1504.

[13] Nicolo Cesa-Bianchi Peter Auer and Claudio Gentile. "Adaptive and Self-Confident On-Line Learning Algorithms". In: *J. Comput. Syst. Sci.* 64 (2002), pp. 48–75. URL: http://www.mit.edu/~6.454/www_fall_2001/shaas/universal_portfolios.pdf.

[14] Kamalika Chaudhuri, Yoav Freund, and Daniel J. Hsu. "A parameter-free hedging algorithm". In: *CoRR* abs/0903.2851 (2009). arXiv: 0903.2851. URL: http://arxiv.org/abs/0903.2851.

[15] Vitaly Kuznetsov and Mehryar Mohri. "Time series prediction and online learning". In: *29th Annual Conference on Learning Theory*. Ed. by Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir. Vol. 49. Proceedings of Machine Learning Research. Columbia University, New York, New York, USA: PMLR, 23–26 Jun 2016, pp. 1190–1213. URL: http://proceedings.mlr.press/v49/kuznetsov16.html.

[16] Yunmei Chen and Xiaojing Ye. *Projection Onto A Simplex*. 2011. arXiv: 1101.6081 [math.OC].

[17] Elad Hazan, Alexander Rakhlin, and Peter L. Bartlett. "Adaptive Online Gradient Descent". In: *Advances in Neural Information Processing Systems 20*. Ed. by J. C. Platt, D. Koller, Y. Singer, et al. Curran Associates, Inc., 2008, pp. 65–72. URL: http://papers.nips.cc/paper/3319-adaptive-online-gradient-descent.pdf.

[18] In: (). URL: https://www.tensorflow.org/tutorials/structured_data/time_series.

[19] In: (). URL: https://machinelearningmastery.com/instability-online-learning-stateful-lstm-time-series-forecasting.

[20] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.