Stock Price Predictor

Juan Javier Arosemena

Introduction

As the knowledge and techniques surrounding machine learning increase, the interest in applying this knowledge to stock data for making predictions is growing as well. The stock market is a composition of buyers and sellers of stocks, which are units for representing partial ownership of a company. These stocks have a specified price which can vary each day and minute, and it is affected by unpredictable factors such as politics, social trends, the environment, and company-related events. Stock data is information that represents the movement of stock prices for given companies (or market indexes such as S&P 500, NASDAQ-100) for each day that the stock market operates.

Stock data usually has 7 main data fields per day:

- Date: the date of the stock data for that day
- Open: the price of the first stock transaction made after market opens
- High: the highest price of the stock
- Low: the lowest price of the stock
- Close: the price of the first stock transaction made before market closes
- Volume: the number of stocks traded that day
- Adjusted Close: the closing price of a stock after considering corporate actions

Based on this data and derived data, financial analysts can make technical analysis for the direction that the stock prices will take and, therefore, make decisions on buying and selling stocks with the lowest possible risk of loss.

In stock market theory, it is said that the market follows the Efficient Markets Hypothesis, which states that the market "follows a random walk and can be unpredictable based on historical data" (Madge, 2015). This holds true for stock predictions in the short term, however, it is possible to find patterns in stock data in longer periods of time, which in turn means that there is a degree of predictableness in the stock market.

Taking into account the fact that stock data holds complex patterns that can give information to formulate predictions, the application of artificial neural networks (ANNs) to this data seems to be an appealing task for exploiting the potential of Artificial Intelligence (in fact, big hedge funds already use AI for stock predictions).

Project Overview

Problem to solve

For stock traders, market predictions are like a compass pointing treasure; money can be made from them. Therefore, development of models that can take in stock data and return predictions is a thriving force in the artificial intelligence field. More specifically, knowing whether the price of a stock will rise (or fall) is profitable information. Given the fact that the stock market is not 100% unpredictable, the possibility for profiting from its historical data exists. Therefore, analyzing stock data with artificial

intelligence is a possible solution for this problem, and a model's performance can be measured based on the accuracy of the prediction against the real-world data. It is expected that using AI to predict stock data yields at least more than 50% accuracy; in other words, it is expected to work better than a coinflip.

Metrics

The model will be evaluated by its accuracy, that is, the percentage of times that the model predicted a price movement correctly on the testing dataset. Since there will be various models trained on different data, various accuracies will be obtained. These accuracies will be compared to the accuracy of the respective benchmark model for each combination of periods in the datasets. The models to compare to are the ones defined and trained by Saahil Madge in his independent report (Madge, 2015).

Workflow

The project is implemented using Python, specifically using PyTorch for defining the feedforward model. All the work done in this project is made using Jupyter Notebooks. The workflow to follow is:

1. Data retrieval and cleaning
    - The data from the NASDAQ-100 Technology index and all other companies is downloaded from Tiingo.com, IEXtrading.com, and Yahoo Finance and converted to Pandas DataFrames. Each row is identified by its date.
    - The data is cleaned by removing records with missing information. Additionally, index and stock data are trimmed so that they only contain records with the same dates.
2. Feature calculation
    - For each date, the input features are calculated using all different combinations of time periods n = (5, 10, 20, 90, 270 days previous) for both index and stock data. This makes a total of 25 different datasets.
    - For each obtained dataset their labels are assigned. I'm using different timespans for predicted values m = (1, 5, 10, 20, 90, 270 days after), so each data record's label corresponds to either 1 for a price increase or 0 for decrease after m days. By now there will be 5*5 different datasets, each of which are assigned a label for each of the 6 different m's. This means a total of 150 different feedforward model instances to train.
    - Every dataset is then separated into training and testing sets.
3. Model definition and training
    - A unique feedforward model architecture is be defined. The model receives the calculated input features, passes them through one or more hidden layers, and finally yields a single value (0, 1).
    - The model is trained separately on every dataset.
4. Results and Benchmarking
    - Once the models are trained, the test datasets are fed to their respective models and accuracy is calculated.
    - Finally, these accuracies are compared to the accuracies obtained by Madge.

Data Exploration

Data Exploration consists of the entire process of finding your data, converting it into data that can be manipulated in code, extracting features from the data, cleaning it, and finally constructing files that contain directly feedable features for an ANN or any machine learning model of choice. For this project the data is composed of two parts: individual stock data and general index data. More specifically, the data is a mix of indicators calculated from stock and index data.

Data Retrieval

Pandas-datareader is the library used for retrieving the stock quotes from the companies listed by the NASDAQ-100 Tech. In order to download the data the source must be specified, which is Tiingo.com and IEXtrading.com for the stock quotes that Tiingo does not have. For using any of these sources an API key linked to a respective account must be passed into the method call that downloads the data. For this case free tier accounts are used, and so the data retrieval is limited to up to five years of past data from the day of download.

Yahoo Finance is the source used for the NASDAQ-100 Tech Index (^NDXT). Since Pandas-datareader no longer supports Yahoo Finance as a source, the Yfinance library was used instead. This is an open source python library developed by Ran Aroussi and distributed in Pypi.org. The index data retrieved from Yfinance has a date limit longer than five years, therefore five years is the timespan limit for both stock and index data to process.

All data is downloaded and directly transformed into a Pandas DataFrame. Immediately after downloading, the data is saved into .csv files and is now considered raw data for the purposes of this project.

Data Preprocessing

In order to manipulate the raw data, it is necessary to give it proper structure. Each company's stock data is held in a DataFrame inside a list variable, while the index data is a single DataFrame variable. For each stock in the list, their date field is formatted and converted to each DataFrame's index. Unnecessary information such as dividends and splits is removed from the DataFrames.

A very crucial part for the following data processing is making sure that every DataFrame, both stocks and index, contain the same ranges of data. This is because the final features are a mix of individual stock quotes with index quotes of the same day. To perform this data trimming, the start and end dates must be chosen. The start date is the oldest last date among all quotes and index, and the end date is the newest first date for all data as well. After finding these dates every DataFrame is trimmed to contain quotes only within that time range. Also, DataFrames are reversed to that the newest quotes are first.

Feature Engineering

Now that all the raw data is transformed into explorable data, we can extract and compute information that we want to feed our machine learning model. The features to calculate for stock and index data are the following indicators:

1. Price Momentum Oscillator = TC – PPC
    - TC: today's close
    - PPC: previous period's close

2. Relative Strength Index = 100 – [100/(1 + RS)]
    - RS: average of x days up-closes divided by average of x days down-closes

3. Money Flow Index = 100 *(100/(1 + MR))
    - MR = (PositiveMF / NegativeMF)
    - MF = TP * Volume
    - TP: average of high, low, and close prices for a given period. If the current Typical Price is greater than the previous period's, it is considered Positive Money Flow.

4. Exponential Moving Average = [α * TC] + [(1 – α) * YEMA]
    - TC: today's close
    - YEMA: yesterday's exponential moving average
    - α: smoothing factor which is 2/(n+1) where n is the number of days in the period.

5. Stochastic Oscillator = [(CP - LP) / (HP - LP)]*100
    - CP: closing price
    - LP: lowest low price in the period
    - HP: highest high price in the period

6. Moving Average Convergence/Divergence = (n-day EMA) – ((1.5*n)-day EMA)

These features where proposed by Abdul Salam, Emary, and Zawbaa (2018) in their paper for a machine learning model for stock market prediction. Additionally, we stick to these 6 time ranges to use for label calculations as proposed by Madge:

1. One day
2. One week
3. Two weeks
4. One month
5. Four months
6. One year

Since the retrieved data does not contain a closing price change field, we must compute it. A function is defined in the Notebook for every proposed indicator, change, and labeling. Each function receives 2 parameters:

1. The DataFrame containing stock/index quotes in descending date order.
2. The period to use to calculate each indicator, label, or closing price change.

For calculating the features, I am using the following periods:

1. One week
2. Two weeks
3. One month
4. Four months
5. One year

A for loop is performed for every possible pair of feature periods (5*5 = 25 in total) to calculate features. In each iteration, the index features are calculated: change, MACD, SO, EMA, RSI, and PMO. MFI is not calculated for index data because it requires volume. Next, the same features plus MFI and each of the 6 different labels are calculated for every stock DataFrame. Every feature gets normalized in every DataFrame as well. Indicators that range from 0 to 100 are scaled down to 0 and 1, while all other indicators are scaled using a MinMaxScaler. Finally, every index and stock DataFrame is saved into csv files as processed data.

This segment of the project is the most time and resource consuming, being capable of taking hours or days to compute.

Final Data Preparation

All data calculations are finished and ready to be built into feedable features. The processed data must be separated into 150 different datasets that belong each to a different model instance to train. We produce one test and one train dataset for each of the 150 models to be trained, and each record (each stock market day) of each dataset contains the following structure:

1. Label
2. Stock PMO
3. Stock EMA
4. Stock MACD
5. Stock RSI
6. Stock MFI
7. Stock SO
8. Index PMO
9. Index EMA
10. Index MACD
11. Index RSI
12. Index SO

The proportion for train and test data separation is 2:1, being the oldest 66.67% of historical stock data the train portion and the newest 33.33% the test portion. Each pair of test and training data files is saved into 150 separate folders that correspond to each model to train and validate.

Training

The modeling phase of the machine learning workflow consists in defining the models to be trained, train the created models, and subsequently test each model's accuracy. For this project I am using Amazon Sagemaker for the training, testing, and deployment of the defined models. However, I am using my API keys to execute all the code locally in my machine.

In order to work with Amazon Sagemaker locally it is necessary to define an IAM Role in the AWS console. This role gives all Sagemaker permissions to the model estimators to create. Along with the AWS Access key and AWS Secret key, the IAM role identifier is stored as an environment variable. The first thing to do after this is uploading all the training data to an S3 bucket.

Model Definition

The machine learning structure to use in this project is Artificial Neural Network (ANN) because of its capabilities for binary classification. ANNs have a structure of three parts: the input layer, the hidden layer(s), and the output layer. Each layer contains several neurons, and each neuron contains a weight (a number). Every neuron from every layer is connected to every other neuron in the adjacent layer(s). The ANN makes its calculations by passing each feature into each of the input neurons, multiplies them through a series of weights to the next layer and so on until the output layer. For binary classifiers, the output layer is only one neuron that returns a value between 0 and 1. This value represents the probability of a certain stock price to go up.

At first, this ANN yields random results, but after each pass or set off passes of training data the model weights are modified so that they give results closer to the expected true value. Training a model on an entire training set is called an epoch, and the more epochs a model trains, the more accurate its predictions are. However, training a model too much on the same training data can overfit the model, so a reasonable number of epochs must be selected. For this case, every model is trained for 10 epochs.

Two important parameters for defining an ANN model to be trained are the criterion and the optimizer, both of which are techniques for the backpropagation of the ANN. For criterion I am using the Binary Cross Entropy Loss, and for the optimizer I am using Adam optimizer.

Under the "source/" directory is the file "model.py", which contains the definition for a class named BinaryClassifier. This class defines the base ANN model for this project which has the following structure:

1. Three parameters need to be passed to the model:
    - input_features: the number of neurons to create for input (11 in this case)
    - hidden_dim: a parameter used to define the ANN hidden layers.
    - output_dim: the number of neurons in the final layer of the ANN. For a binary classifier this is 1, and the result ranges from [0,1].
2. The number of neurons in the 4 hidden layers of the model are defined as:
    - hidden_dim
    - 2 * hidden_dim
    - 3 * hidden_dim
    - hidden_dim
3. The forward pass of the model
    - Input layer -> Linear transform to the first hidden layer
    - Passed into Rectifier Linear Unit function
    - Dropout layer (for training only)
    - Repeat the above steps until the final hidden layer...
    - Last hidden layer -> Linear transform to the output layer
    - Sigmoid Activation Function -> Result

Model Training

Under "source/" there is a file named "train.py", which contains the structure for a PyTorch entry point. This is necesssary for creating estimators through Sagemaker. Both "model.py" and "train.py" are files borrowed from the Udacity Machine Learning Nanodegree but adapted to fit this project.

A for loop is made for every combination of two periods for feature calculations plus the prediction ranges used for labeling. This yields a total of 150 iterations. In each iteration a PyTorch estimator is defined and fit to the training data. Every 10 iterations the Sagemaker session is made to wait for completion since I have a limited number of instances I can use at a time.

It is worth noting that I obtained a ThrottlingError while executing this piece of code at first because I trained more than 10 instances at a time. This only paused the entire process, so I could continue on after the error instead of restarting all training instances.

Model Evaluation

For evaluation each training job created is deployed in AWS. For each estimator, a predictor endpoint is created briefly to be sent data to make predictions. For each predictor, their respective test datasets are passed. The endpoint for the predictor is then deleted. Then, accuracy calculations are made against the labeled test datasets, they are printed and stored into .txt files.

While executing this code the first time I got a ModelError because of low instance capacity. The initial instance I used for this part was "ml.m4.xlarge", but later changed it to "ml.m4.2xlarge" and the deployments continued correctly.

Results & Benchmarking

Now the 150 models are trained and evaluated, and we can identify those that performed the best and compare them to Saahil Madge's models in his report. The first step in this final part is to get both my results and Madge's results into code. For the latter I hardcoded the results from the paper into the Jupyter Notebook, and read my results from the saved files.

Top Performers

The top ten best predicting models trained have accuracies higher than 85%, and they all outperform Madge's models with the same data parameters. The field *accuracy* and *madge_mean* in table 1 represent each model's obtained accuracy and the accuracy mean obtained by Madge, respectively.

Also, Madge's top ten best performing models are all outperformed by their counterpart in this project, as seen in table 2. Figure 1 shows a line graph of my obtained accuracies along every model compared to Madge's accuracies. Only few of my trained models are outperformed by their respective model in Madge's paper. There is a positive trend between the models' accuracies as parameters grow. However, it can be noticed that the accuracies diverge for longer prediction periods; Madge's accuracies drop while our models' accuracies grow.

Figure 3 shows every model comparison at a closer level. It shows the same trend between both data than in figure 1 but at a more granular level. From these results, 83.33% of my trained models outperform Madge's models.

Effective Accuracies

There is something to consider, though. If a model's accuracy surpasses 50%, it means that it can predict the stock trend better than a coinflip, as Madge says (Madge, 2015). However, if a model's accuracy is below 50% it means that most of its predictions will be incorrect. Since a model's prediction is a binary value (increase or decrease stock price), one value is correct and the other one is incorrect. If we know that a model will give mostly wrong predictions, we can assume the opposite result to be true most

of the time. We call the accuracy of the results obtained this way Effective Accuracy. Therefore, if we reverse the result of those models with prediction accuracies below 50%, we can treat their effective accuracies as:

$$acc_{eff} = 0.5 + (|0.5 - acc_{real}|)$$

where acc_{eff} is the effective accuracy of any model and acc_{real} is the real tested accuracy of that model. In fact, this formula can be applied to know any model's effective accuracy, regardless of its real accuracy.

With this perspective, we can reanalyze the data in Madge's results against the obtained results. The effective accuracies are shown in Figures 2 and 4. Just like with real accuracies, the majority of my models' effective accuracies are higher than Madge's, and even so that 84.0% of my trained models effective accuracy outperform Madge's models.

Final Analysis

Even though Madge's accuracies were overall increased for calculating effective accuracy, they still do not outperform my trained models. Surprisingly, many of my models obtained accuracies above 80%, which is itself a good number to keep exploring. It is worth noting that a good model is also defined by its prediction range, so a shorter time range is more valuable than longer one. Most of the best performing models are those that predict trend a year into the future, nonetheless, some high performing models predict four months and even one month into the future. Additionally, it seems that the period parameter for the index part of the features affects a model's accuracy strongly. The several spikes in figures 1 and 3 are due to the increasing n2 parameter along the models.

These may be promising results, however, further experimentation is needed. The results I obtained are not statistically comparable to Madge's results because only one instance per model to was trained, while Madge conducted several training instances and obtained a mean of the results.

Future Work

Future work for this project will include several pivot points. First, future experimentation will focus on the models that performed the best, prioritizing those with the smallest prediction time range. Additionally, the machine learning algorithm used by Saahil Madge, Support Vector Machine, will be implemented; It is known to be very experimented on with stock data predictions. Feature experimentation will be necessary as well and may include more and mixed stock data calculations. Finally, the models will be tested against real time stock data to prove their tested accuracies.

References

1. Madge, S. (2015) *Predicting Stock Price Direction using Support Vector Machines.* Independent Work Report Spring 2015. Computer Science Department of Princeton University. Available at: https://www.cs.princeton.edu/sites/default/files/uploads/saahil_madge.pdf
2. Abdul Salam, M., Emary, E., Zawbaa, H. (2018). *A Hybrid Moth-Flame Optimization and Extreme Learning Machine Model for Financial Forecasting.* Available at: https://www.researchgate.net/publication/324029737_A_Hybrid_Moth-Flame_Optimization_and_Extreme_Learning_Machine_Model_for_Financial_Forecasting

Annex

Table 1: Top ten performers of the 150 models trained, compared to the analogous model in Madge's paper.

	model	prediction_range	n1	n2	accuracy	madge_mean
141	270-90-10	270	90	10	0.898103	0.449840
140	270-90-5	270	90	5	0.898103	0.444788
138	270-20-90	270	20	90	0.898103	0.413345
128	270-5-90	270	5	90	0.898103	0.389645
142	270-90-20	270	90	20	0.898103	0.461921
133	270-10-90	270	10	90	0.898103	0.399003
143	270-90-90	270	90	90	0.898103	0.432547
144	270-90-270	270	90	270	0.874204	0.526434
139	270-20-270	270	20	270	0.874204	0.521269
129	270-5-270	270	5	270	0.874204	0.517076

Table 2: Madge's top ten performing models compared to my models with the same parameters.

	model	prediction_range	n1	n2	accuracy	madge_mean
145	270-270-5	270	270	5	0.874204	0.691244
146	270-270-10	270	270	10	0.874204	0.687763
147	270-270-20	270	270	20	0.874204	0.675214
148	270-270-90	270	270	90	0.874204	0.674632
120	90-270-5	90	270	5	0.843332	0.614993
121	90-270-10	90	270	10	0.843332	0.605926
122	90-270-20	90	270	20	0.843332	0.604149
123	90-270-90	90	270	90	0.843332	0.598991
96	20-270-10	20	270	10	0.682513	0.578369
114	90-20-270	90	20	270	0.843332	0.577826

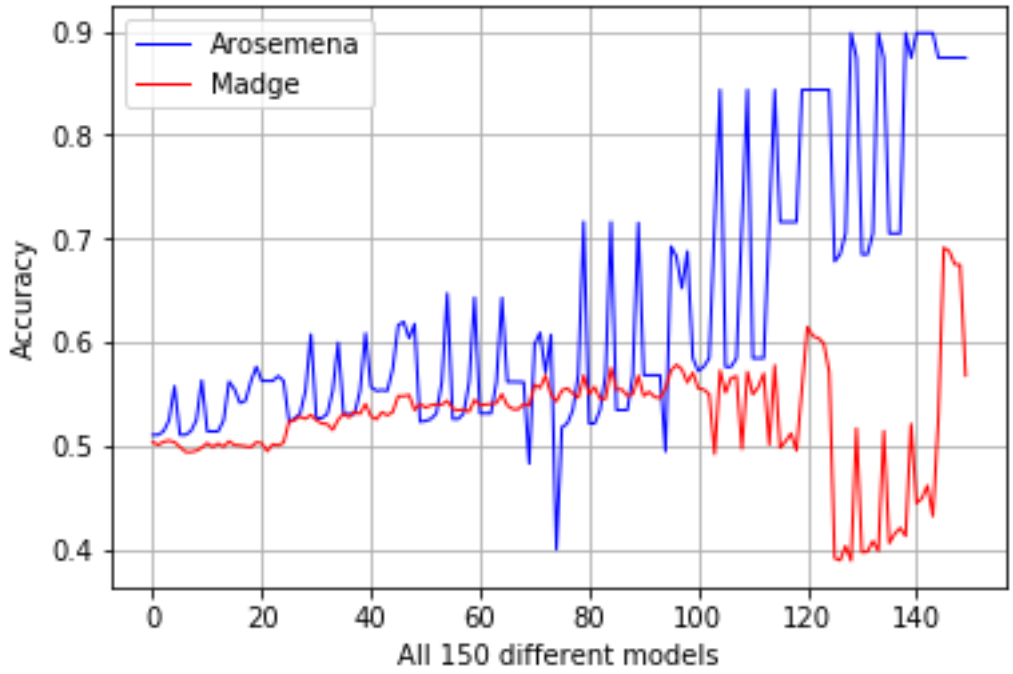

Fig. 1: Line graph of the obtained accuracies vs. Madge's accuracies.

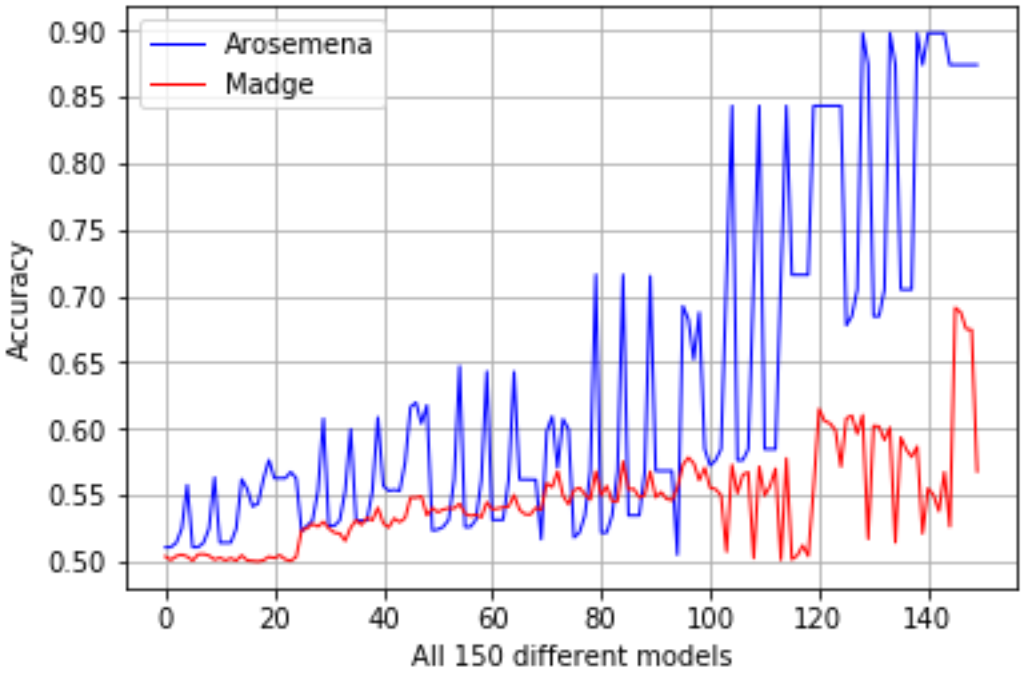

Fig. 2: Line graph of effective accuracies.

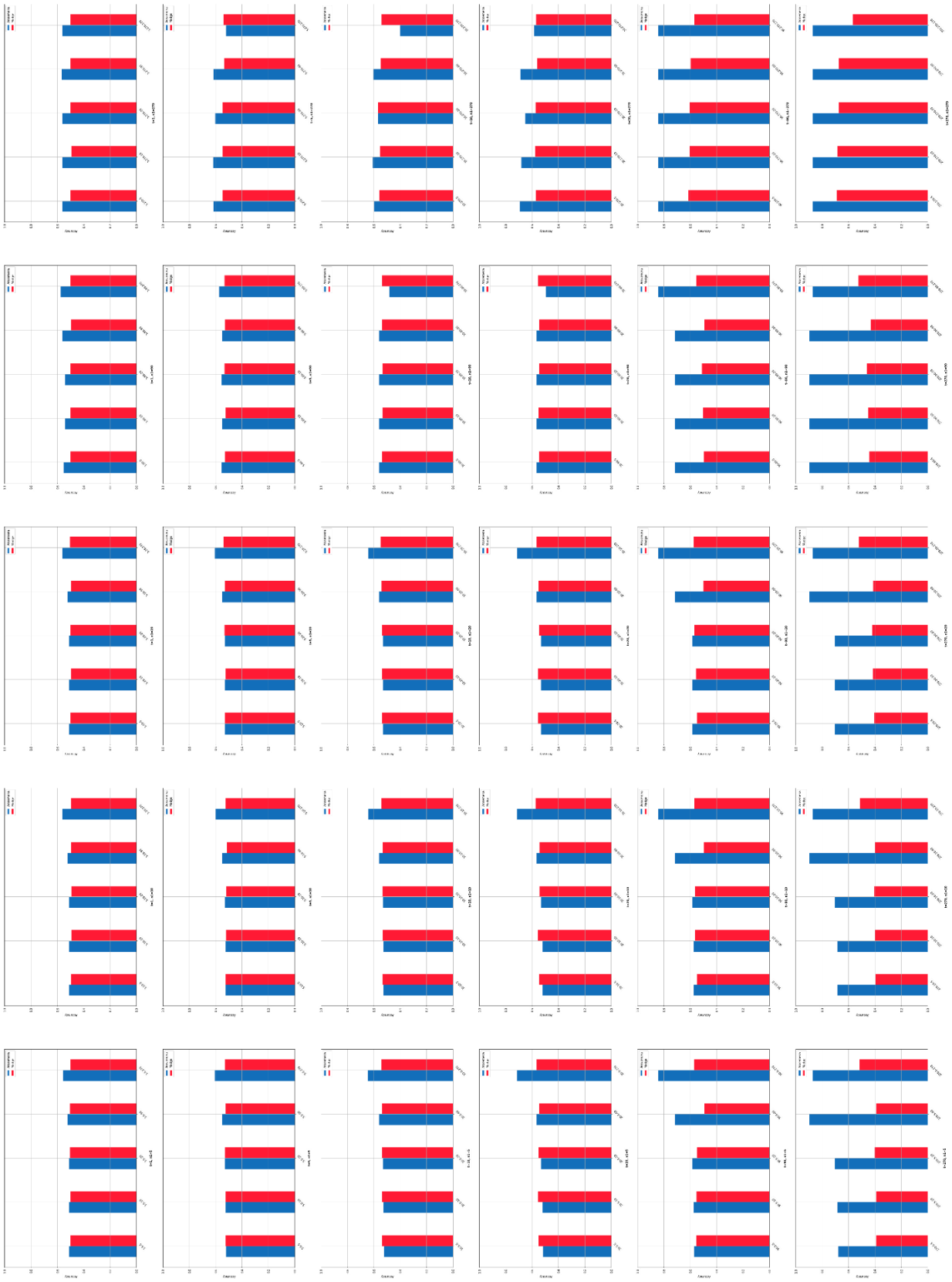

Fig. 3: 150 pairs of bar graphs for Madge's (red) and my (blue) accuracies for every model.

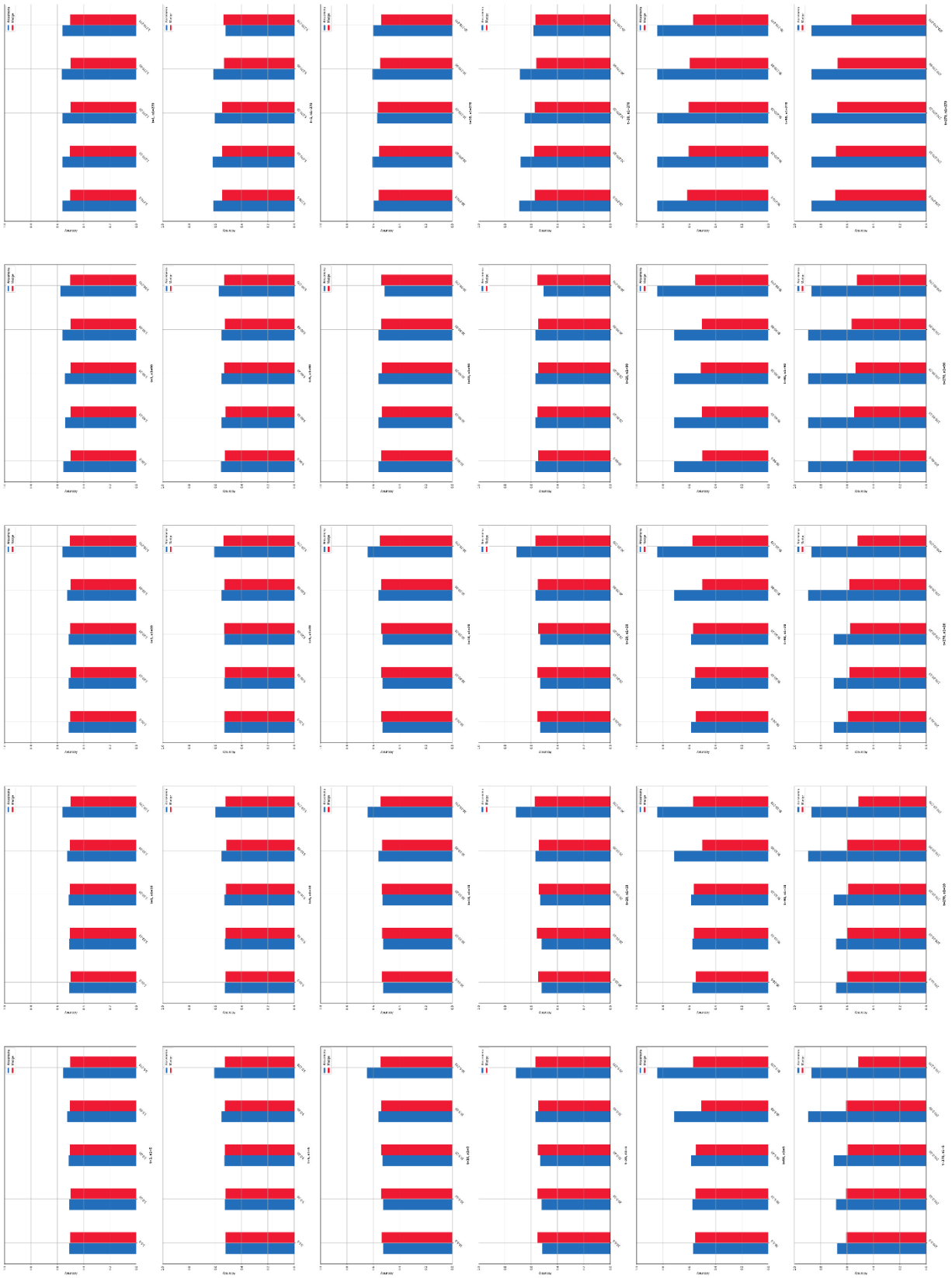

Fig. 4: 150 pairs of bar graphs for Madge's (red) and my (blue) effective accuracies for every model.