Lorenzo

Stock Market Technical Analysis

Pattern Recognition on Price Data using K-Nearest Neighbors

02 February 2018

Purpose:

The final goal in producing a predictive model of stock price action is usually to integrate it into a deployable running trading strategy, which consists of at least one rule or model. However the goal of this research project was not to invent an entire trading strategy but evaluate a single model's ability (correlation) at predicting one-day price-action (percent change) for each stock in the S&P 500, where the model 'trains' using only the price data for all the stock in the S&P 500. The observed model uses pattern recognition with a similarity function of two price series to find the closest fixed-length price series patterns throughout a period of the market's history, then predicts the following day's price-action by averaging of each of the outcomes following the matched price series pattern. This algorithm of finding the most similar examples from a labeled dataset and averaging their labels, known as k-nearest neighbors, is a form of instance based learning and is computationally expensive but can offer certain benefits like simplicity and transparency, which can be crucial in distinguishing certain patterns inside the noisy and chaotic stock market. In this study I sought to find predictive power, measured by correlation, between the 24-hour price-action of the stocks in the S&P 500 and the respective output from the k-nearest neighbors algorithm, which has trained only using price history data from stocks in the S&P 500.

Method:

In order to separate data retrieval from pre-processing and evaluation StocksKNN.py executes all the computations responsible for run-time preprocessing and evaluation, after the data has once been initially downloaded from Yahoo finance. The class StocksKNN has three methods to train, validate, and test which work on an arbitrarily length of price data given. The class StocksKNN also contains the helper function predict(), which returns the outputs queried by the KNN algorithm, given an array of inputs from one or more price series (after normalization).

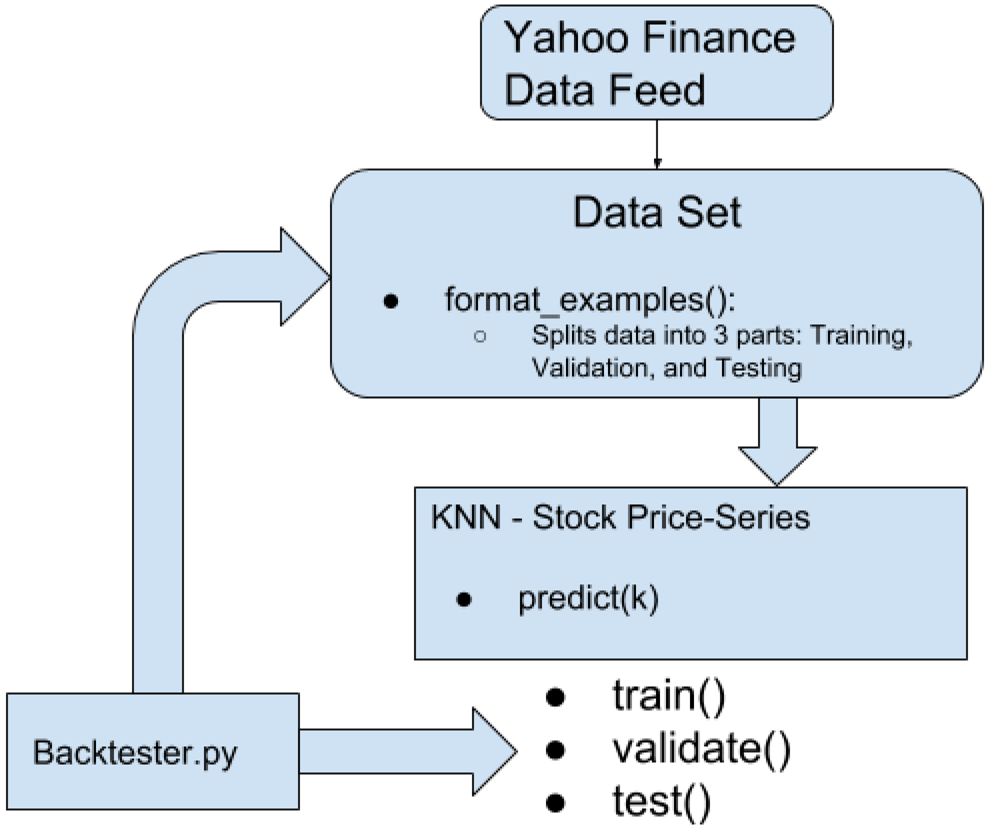
Once pricing data for every stock in the S&P 500 has been downloaded from https://finance.yahoo.com/ between the dates 2017/11/16 -2018-02-05, the data must be preprocessed. In order for the KNN model to compare the similarity, or distance, of two stocks' price series the following steps must be made to normalize each example's price series:

1. Subtract the 'present' day's price from every previous day's price for an example (the 'present' day being the day that precedes the 'outcome' for that respective example). For example, if a stock has risen constantly over n-days, then it's example-vector will have all-negative values corresponding to each day that is below the current day's price.
2. Then, normalize the example vector into percent change by dividing each day's element by the price at the present day.

The label, or outcome, for each example would be the percent change of the day following the price series observation.

Using this transformation to represent price-action over a fixed-length period for any given stock, the KNN can compares\ price series similarity by taking the euclidean distance of the two price series vectors. That is, it finds the root of the sum of the squared differences between each examples respective-day's percentage change from its last price. Using this measure of similarity between price patterns KNN can identify the most similar examples and use the examples' empirical outcomes to estimate a probability distribution of outcomes. For my experiment I simply measured the mean of the most similar patterns' outcomes, and used that as the KNN model's output, ignoring measures of spread.

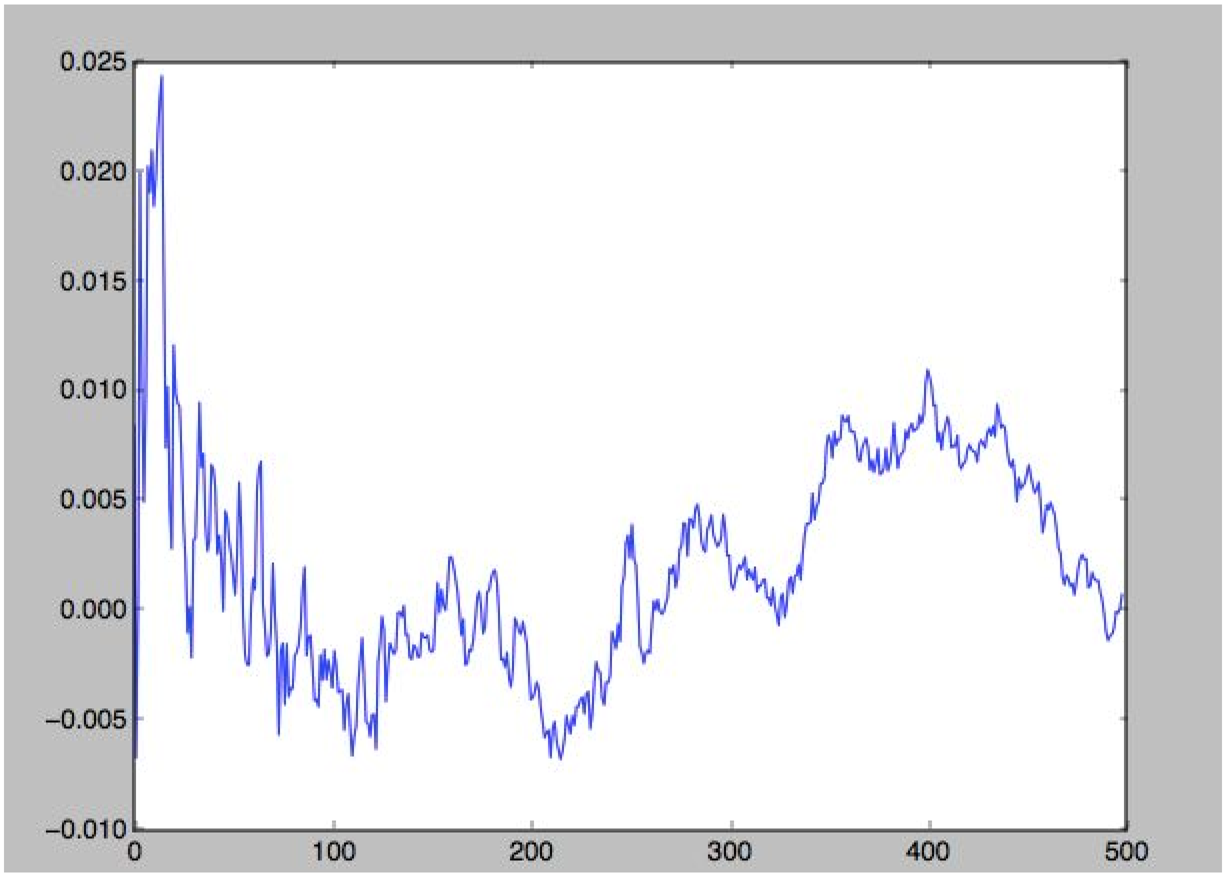
Once the data is partitioned in an 6:1:1 training, validation, testing ratio then the model iterates over every validation example to find the optimal value for K--choosing whichever K that maximizes the output-outcome correlation. Then that K is used to evaluate the algorithm on the testing set, once the new data from the validation set has been added to the KNN model's memory.



Findings:

Testing the KNN model's performance on days 2017/11/16 -2018-02-05 I found a correlation of 0.01976 for sample size 2808 (p=0.1474935) using K = 15.

Here is a graph of the validation performance of each K (x-axis) and its resulting correlation (y-axis).



Further Discussion:

The final goal in producing a predictive model of stock price action is usually to integrate it into a deployable running trading strategy, which consists of at least one rule or model. Typically trading strategies may operate using a combination of simple and complicated models and inputs--such as 'Only buy Tesla stock on sunny days', and 'Minimize diversifiable risk among correlated assets." Deriving models independently of each other is a strategy that tends to work well because it lowers risk of overfitting in the noisy and chaotic data of the stock market. When conflicts between models arise during runtime either another model authority must step in and break the tie, or in particular applications like trading where not every opportunity needs to be classified, most examples may be ignored. In the large universe of tradable instruments, the opportunities presented to the trader are numerous enough that confused/conflicting trading opportunities can be sometimes overlooked entirely in favor of unanimously agreed upon opportunities where all models' predictions align. In this way a selective-ensemble of predictors is made even more powerful than the best of all its models (assuming each model's performance is random, at worst) simply by limiting the domain of its predictions to the all the examples in which all its component models are harmonious.

With the ultimate goal in mind of setting up multiple "weak" predictors into an actionable trading strategy I hypothesized that I would want to use k-nearest neighbors because its mechanism of prediction provides easily derivable confidence metrics. The standard k-nearest neighbors algorithm works by searching a labeled dataset using a similarity function to rank the similarity of every known example to a given query example (one it is trying to make a prediction on). Then, the k-most similar examples in the training set are referenced and their labels averaged to generate an output. While merely averaging the labels for each of the known examples can be powerful, it neglects to consider the spread (which could be thought of as the "certainty") of the distribution of observed outcomes, as well as the similarity between the queried example and each of the stored examples (which could be thought of as the "discrepancy" between the input and the query). Both these metrics are intuitively useful at determining a models confidence estimate for its prediction, additional features in a modified KNN predictions may be useful in retaining a large domain of stock opportunities, while utilizing as many models synergistically as possible.