CS51 Project: Draft Specification

Project Name

FX Predictor - an application of machine learning toward financial markets

Group member

Masakazu Tanami <tanami@g.harvard.edu>
Ben Guild <ben@benguild.com>
Ethan Brooks <ethanabrooks@gmail.com>
Kaleem Abdullah <kaleem.abdullah@gmail.com>

Brief Overview

Many people use historical charts for analyzing financial markets to find patterns to predict future markets. This is because we, as human beings, don't really change the way we think; that is to say, "History repeats itself." Our product makes use of the power of computation, aiming to help such people find patterns to predict future markets.

As we use machine learning, it allows us to exhaustively search through the historical data. Let's say we have 10 years of 1 minute candle data (Open, High, Low, Close), which counts roughly 4 million candles. It's far beyond a human trader's ability to traverse millions of candle data every minute, so our product would be a great asset.

When run on a currency pair, the system will return an up/down (higher or lower than current FX rate) prediction and its probability (USD/JPY in 10min: UP (78%)). The system receives market data and automatically updates its internal data structure, returning a result every minute until terminated. This is very useful when you invest in a binary option.

In terms of functionality, the program loads historical data and builds a certain data structure, analyzing current market data with pattern-matching algorithms to predict future FX rates.

Basic concepts

Take the following price history of 10 minutes as an example:, for example, [120;122;119;115;117;121;123;125;124;128]. The program creates a certain data structure and tries to find several similar paths by applying the Naive Bayes Classifier Algorithm. Then calculating weighted average prices in 10min for prediction.

Feature List

Our software will predict future FX prices, let's say up to 3 min later. How to do this? Well, the system will have 2 major components, Pattern Match and Prediction. We will mostly woking on pattern match algorithms and data structures.

[Pattern Match Component]

Primary List

- Data creation. Take a csv file to return a list of candles.
- Candle equality checker. Compare twocandles to return bool.
- Filter function for faster equality check.
- Path similarity checker. Compare two paths to return a similarity score of int.
- Data structure builder. Take a list of candles to return a list of paths.
- Scan function. Traverse to choose the most similar paths (i.e. highest similarity score) as a list

Secondary List

- Real time data feed.

The Pattern Match component searches for the most similar historical price path to the current price path. Here, what we call the current price path on USDJPY for 3 min would look like this.

```
   Time    | Open | High | Low | Close
 3 min ago  120    122    119    120
 2 min ago  120    122    118    120
 1 min ago  120    122    117    120
```

Here is a matching path (it has exact the same data except time).

```
   Time    | Open | High | Low | Close
 6 min ago  120    122    119    120
 5 min ago  120    122    118    120
 4 min ago  120    122    117    120
```

Alternately, under the Naive Bayes classification, a path with a different order such as the reverse order of the current path would be considered similar or matched (to be explained later).

```
   Time    | Open | High | Low | Close
 9 min ago  120    122    117    120
 8 min ago  120    122    118    120
 7 min ago  120    122    119    120
```

By convention, we call a set of this type of data (Open, High, Low, Close) a “candle.”
http://en.wikipedia.org/wiki/Candlestick_chart

Candle Equality

Now that we understand the basic strategy of predicting FX. Let's take a look at the actual candle data.

```
    Date    |    Time     |  Open   |  High   |  Low   | Close
02.03.2015 15:00:00.000  119.787  119.891  119.76  119.887
```

Then suppose if 1 min later a candle looks as follows, then compared with the one above, are those equal?

```
    Date    |    Time     |  Open   |  High   |  Low   | Close
02.03.2015 15:01:00.000  119.786  119.891  119.76  119.887
```

The current answer is no. Because the open price is different by 0.001. Well, in 10 years of 1 min data (3,744,000 candles), we may not find an exact match . So how we can handle this issue?

Filter Function

Solution1: We can take differences from the open price.

```
    Date    |    Time     |  Open   |  High   |  Low   | Close
02.03.2015 15:00:00.000 000.000  000.104  -000.27 000.100
```

We may multiply these by 1000 to get integer value for faster operations (maybe).

```
    Date    |    Time     |  Open   |  High   |  Low   | Close
02.03.2015 15:00:00.000     0       104      -27     100
```

Solution2: We can ignore the third decimal place and multiply by 100 in order to loosen the requirements for equality.

```
    Date    |    Time     |  Open   |  High   |  Low   | Close
02.03.2015 15:00:00.000     0       10       -2      10
```

Solution3: We can only use a difference of open and close and multiply by 1000. Thus, if Close >= 0 it means 'UP' from open price and 'Down' otherwise.

```
    Date    |    Time     |  Open   |  High   |  Low   | Close
02.03.2015 15:00:00.000     0        0        0      100
```

Path Similarity

(Primary focus)

Naive-Bayes Classification Algorithm

Reference: The chapter 13 Text classification & Naive Bayes.

http://nlp.stanford.edu/IR-book/information-retrieval-book.html

(Secondary focus)

Something similar to page ranking algorithm or Google Suggest algorithm

http://infolab.stanford.edu/~backrub/google.html

http://www.google.co.jp/patents/US8438142

http://www.google.co.jp/patents/US8321403

Naive Bayes for Document Classification

Before we actually apply the Naive Byes Classification algorithm, let us explain generally how it works for document classifications.

We have two groups of sentences here.

A: Greg is awesome
B: I like CS51

Now if we have a new document(sentence), which group is considered more similar?

C: CS51 is awesome

The algorithm will count the number of words appeared in each group to calculate a score of relevance.

A: Greg is awesome (is, awesome -> score: 2)
B: I like CS51 (CS51 -> score: 1)

According to its relevance score, Group A is considered the more similar. Note that we just summed up the number of appearance, but in more general algorithm, it will multiply the probability of appearance. We may discuss this later but we'd use this algorithm in this document.

Naive Bayes for FX Path Classification

Let's apply the rule above to FX. We can treat a 1 min candle as a word, and a group of candles (path) as a sentence.

We have the most recent path as a base of comparison. (training data)

Current Path:

```
3 min ago  120    122    119    120
2 min ago  120    122    118    120
1 min ago  120    122    117    120
```

And we have 3 paths from historical candle data to be classified.

Path A:

```
6 min ago  120    122    119    120 (Equal to a candle in 3 min ago)
5 min ago  120    122    118    120 (Equal to a candle in 2 min ago)
4 min ago  120    122    117    120 (Equal to a candle as 1 min ago)
```

Path B:

```
9 min ago  120    122    117    120 (Equal to a candle as 1 min ago)
8 min ago  120    122    118    120 (Equal to a candle in 2 min ago)
7 min ago  120    122    119    120 (Equal to a candle in 3 min ago)
```

Path C:

```
12 min ago  120    120    117    120 (Not equal)
11 min ago  120    122    118    120 (Equal to a candle in 2 min ago)
10 min ago  120    122    119    120 (Equal to a candle in 3 min ago)
```

Finally, if we try to find the most similar paths to the current path, both A and B have the same highest score 3, and are therefore considered "matched."

Sliding window paths creation

In order to exhaustively search through the historical data, we use sliding window approach to create paths for classification. In the example above, the current path is (1-3min), then the first path to be compared is (2-4min), the second path to be compared is (3-5min), ... (10-12min). In general, if we have n candle data and compare m minutes of path, then the number of paths to be compared is n-m. Note that each window has m-1 numbers of overlapping, so we have to take that into consideration.

Mathematical Foundation

We use Naive Bayes for pattern match in order to calculate similarity between a path from historical data and a current path (training data).

This is the basic formula behind the Naive Bayes Classification algorithm from our reference textbook (Chapter 13.2, P.258):

$$P(c|d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k|c)$$

This is what these variables equal in our application:

- c is the event that a path is categorized to a category path. In our case, a category path is a current path.
- d is a collection of training paths. In our case, there is one training path (current path) with candles (in our example, 3 candles but can be 7200 or arbitrary number) in it.
- t_k is a single piece (a candle) in a training path.
- n_d is the number of candles in d.
- $P(c|d)$ is the probability that a path is categorized to a training path. (similarity)
- $P(c)$ is the percent of each category path out of the total number of training paths. In our case, there is one category out of total number of training paths, so $p(c) = 1$.
- $P(t_k|c)$ is the percent of candles in a path that matches candles in a category path (= a current path). In our case, total number of candle is always the same, so we can only count up the matched candles by comparing a path and a current path.

So a path with high probability of P(c|d) is considered similar to a current path.
We could set a threshold to determine if a path is matched or not.

Our scan function is collecting all the paths which is matched to a current path. (This works like spam filter for millions of emails)
After that, our prediction will be made based on these matched paths.

Data Structure
We think this is one of our core developments as well as path similarity.

As to the whole data structure, I picked a list as a primary focus because we check at least once for every single path. We may find a better data structure as a secondary focus.

As to the path data structure, each path is either a ListQueue, a TreeQueue, a BinaryHeap, a BinomialHeap, or a HashTable. We wish to compare these for performance.

We initially assume a path has 3-10 candles, but it should be able to have 60(hour) or 1440(day) or 7200(week) candles. We will use a module to implement this.

(primary focus)
1. List of ListQueue (paths)
2. List of TreeQueue (paths)
3. List of BinaryHeap (paths)

(secondary focus)
4. List of BinomialHeap (paths) (http://en.wikipedia.org/wiki/Binomial_heap)
5. List of HashTable (paths)

[Prediction Component]

Primary List
- Calculate an average path from a scanned list.
- Get result (Up/Down) and certainty in percentage.
- Print a result every minute until the program is terminated.

Secondary List
- Mount our system to the dedicated website.
- Show relevant market charts so investors can visually see the difference.

Once we find the similar paths, we would be able to predict the future!

The output format looks like this.
(Currency pair) in (n)min: (UP/DOWN) (Certainty in %) - Average Path (1m: xxx, 2m: xxx, 3m: xxx)

The future average paths could be calculated by averaging 'close' time of each matched candles.

The certainty represents the level of fitting to the historical path and the strength of being Up or Down. Therefore, it could be calculated by taking an average of (actual score/possible max score) * (number of candles whose close price is higher(or lower) than the current open price/ total num of candles).

In this example, it would conceptually look like this - Avergage [Path A:(3/3) * (3/3), Path B:(3/3) * (3/3)].

So in this example, our software will return

```
USDJPY in 3min: UP (100%) - Average Path (1m: 120, 2m:120, 3m:120)
```

Technical Specification

Our core functionality is a machine learning algorithm for pattern recognition and data structure of a path.
We'd break up those in to four groups and try to distribute other implementation evenly.

Group A

- Data creation. Take a csv file to return a list of candles.
- BinaryHeap
- Print result every minute until the program is terminated.

Group B

- Candle equality checker. Compare two candles to return a bool.
- Filter function for faster equality check.
- Calculate an average path from a scanned list.
- HashTable

Group C

- Path similarity checker. Compare two paths to return a similarity score of int.
- Scan function. Traverse to pick up the most similar paths (i.e. highest similarity score) as a list.
- TreeQueue

Group D

- Naive Bayes Classifier
- ListQueue
- Get result (Up/Down) and certainty in %
- BinomialHeap

To Be Assigned

- Real time data feed.
- Mount our system to the dedicated website
- Show relevant market charts so investors can visually see the difference

Types, signatures or interfaces

```
type candle = {o: float; h: float; l: float; c: float}
type order = Equal | NotEqual

module type COMPARABLE =
sig
  type t
  val compare : t -> t -> order
  val to_string : t -> string
  val generate: unit -> t
  val generate_gt: t -> unit -> t
  val generate_lt: t -> unit -> t
  val generate_between: t -> t -> unit -> t option
end

module type BINTREE =
sig
  type elt
  type tree
  val empty : tree
  val search : elt -> tree -> bool
  val insert : elt -> tree -> tree
  val delete : elt -> tree -> tree
  val getmin : tree -> elt
  val getmax : tree -> elt
  val run_tests : unit -> unit
end

module type PRIOQUEUE =
sig
  type elt
  type queue
  val empty : queue
  val is_empty : queue -> bool
  val add : elt -> queue -> queue
  val take : queue -> elt * queue
  val run_tests : unit -> unit
end

module type HASHTABLE =
sig
  type elt
  type hashtable
  val empty : hashtable
  val search : elt -> hashtable -> bool
  val insert : elt -> hashtable -> hashtable
  val delete : elt -> hashtable -> hashtable
  val run_tests : unit -> unit
end
```

Next Steps

The development could be broken down into three phases (one basic algorithm implementation, more algorithms implementation, front-end (web or mobile) development). The first phase might be enough, but here's our initial plan.

[Phase 1] Research on Naive Bayes algorithm, discussion of a basic strategy, discussion of data structures, module design, functionality implementation, testing using historical data.

[Phase 2] Apply the system to real-time market. Research on google suggest, Facebook suggest, etc. Key point is scalability. There will be so many variations of algorithms from just parametric different ones to conceptually different ones. We should be able to add on as much as possible, so we should be able to compare those results.

[Phase 3 (optional)] Some of us might be familiar to front-end development so we can visually play around software through web app or mobile app.