Если вы видите что-то необычное, просто сообщите мне.

Машинное обучение Golang

Вычисление простых статических свойств

Статистическое обучение это ветвь применения статистики которая связана с машинным обучением.

Машинное обучение, которое тесно связано с вычислительной статистикой, это часть информатики которая пробует изучить данные и на их основе делать предсказания о их поведении без специального программирования.

В этой статье, мы собираемся изучить как сосчитать базовые статистические свойства, такие как среднее значение, минимальное и максимальное значение примера, медианное значение, а также дисперсию примера. Эти значения дадут вам отличное понимание вашего примера без среёзного погружения в детали. Однако, общие значения которые описывает пример, могут легко обмануть вас, заставив вас поверить, что вы хорошо знаете пример, и без них.

Все эти статистические свойства, будут посчитаны в `stats.go`, который будет представлен в пяти частях. Каждая строка входного файла содержит одно число, которое значит, что входной файл читается построчно. Неправильный ввод будет проигнорирован без каких либо предупредительных сообщений.

Ввод будет сохранен в срезе, чтобы можно было использовать отдельную функцию для подсчета каждого свойства. Так же, увидим, значения среза будут отсортированны перед обработкой.

```
package main

import (
  "bufio"
  "flag"
  "fmt"
  "io"
  "math"
  "os"
  "sort"
  "strconv"
  "strings"
)


func min(x []float64) float64 {
  return x[0]
}

func max(x []float64) float64 {
  return x[len(x)-1]
}


func meanValue(x []float64) float64 {
 sum := float64(0)
 for _, v := range x {
   sum = sum + v
 }
  return sum / float64(len(x))
}


func medianValue(x []float64) float64 {

 length := len(x)
```

```
 if length%2 == 1 {
 // Odd
  return x[(length-1)/2]
 } else {
  // Even
   return (x[length/2] + x[(length/2)-1]) / 2
 }
  return 0
}


func variance(x []float64) float64 {

 mean := meanValue(x)
 sum := float64(0)

 for _, v := range x {
    sum = sum + (v-mean)*(v-mean)
 }
  return sum / float64(len(x))
}


func main() {

  flag.Parse()

  if len(flag.Args()) == 0 {

  fmt.Printf("usage: stats filename\n")
   return
 }

  data := make([]float64, 0)
  file := flag.Args()[0]

 f, err := os.Open(file)
  if err != nil {
   fmt.Println(err)
     return
```

```
 }

defer f.Close()
```

Регрессия

Регрессия это статистический метод для расчета связи между переменными. Эта часть реализует линейную регрессию, которая наиболее популяная и наиболее простая техника, а так же наилучший способ понять наши данные. Заметим, что регрессия не имеет 100% точность, даже если вы использвали многочлен высшего порядка(нелинейный). Цель регрессия, как и в большинстве ML техник - найти достаточно хорошую технику а не лучшую из лучших.

Линейная регрессия. За этой регрессией прячется следующее: вы берете модель ваших данных используя уравнение первой степени. его можно представить как `y = a x + b`.

Есть множество методов, которые позволяют найти уравнение первого порядка, которое представит модель ваших данных и вычислит `a` и `b`.

Реализация линейной регрессии. Go код, этой части будет сохранен в `regression.go`, который будет представлен в трех частях. Вывод программы будет два числа с плавающей запятой, которые определяют `a` и `b` для ураввнения первого порядка.

Первая часть `regression.go` содержит следующий код:

```
package main



import (
  "encoding/csv"
  "flag"
  "fmt"
  "gonum.org/v1/gonum/stat"
  "os"
  "strconv"
)
```

```
type xy struct {
  x []float64
  y []float64
}


func main() {

  flag.Parse()

  if len(flag.Args()) == 0 {
    fmt.Printf("usage: regression filename\n")
    return
  }

  filename := flag.Args()[0]

  file, err := os.Open(filename)

 if err != nil {

    fmt.Println(err)
   return

 }

defer file.Close()

r := csv.NewReader(file)

records, err := r.ReadAll()

if err != nil {
  fmt.Println(err)
  return
}

size := len(records)
```

```
data := xy{
 x: make([]float64, size),
 y: make([]float64, size),
}

for i, v := range records {
if len(v) != 2 {
fmt.Println("Expected two elements")
continue
}
if s, err := strconv.ParseFloat(v[0], 64); err == nil {
data.y[i] = s
}
if s, err := strconv.ParseFloat(v[1], 64); err == nil {
data.x[i] = s

  }
}

b, a := stat.LinearRegression(data.x, data.y, nil, false)

fmt.Printf("%.4v x + %.4v\n", a, b)
fmt.Printf("a = %.4v b = %.4v\n", a, b)

}
```

Классификация

В статистике и ML, классификация это процесс помещения элементов в существующие наборы которые называются категориями. В ML классификация подразумевает обучение с учителем, в котором набор предполагает содержание верно определенные данные используемые для обучение перед работой с реальными данными.

Очень простой и легко реализуемая метод классификации называется "k-близжайших соседей"(k-NN). Идея метода такова, что мы можем отсортировать данные основываясь на их схожести с другими предметами. "k" в "k-NN" обозначает число соседей которые должны быть включены в решение, которое значит что k это положительное целое. которое

довольно маленькое.

Ввод алгоритма состоит из k-близжайших примеров обучения, в свойстве пространства. Объект классифицируется множеством голосов его соседей, с объектом назначенным классу который в большинстве обобщен среду его k-NN. Если значение k = 1, значит элемент просто назначен классу, который ближе всего согласно расстоянию используемой метрики. Удаленная метрика зависит от данных с которыми работаете. Как пример вам нуно различное расстояние метрик, когда работает со сложными числами и другими, когда работаете в трехмерном пространстве.

```
package main



import (

 "flag"
 "fmt"
 "strconv"
 "github.com/sjwhitworth/golearn/base"
 "github.com/sjwhitworth/golearn/evaluation"
 "github.com/sjwhitworth/golearn/knn"

)



func main() {
 flag.Parse()
 if len(flag.Args()) < 2 {
   fmt.Printf("usage: classify filename k\n")
   return
 }

dataset := flag.Args()[0]

rawData, err := base.ParseCSVToInstances(dataset, false)

 if err != nil {
  fmt.Println(err)
   return
```

```
 }

k, err := strconv.Atoi(flag.Args()[1])

 if err != nil {
  fmt.Println(err)
    return
 }

cls := knn.NewKnnClassifier("euclidean", "linear", k)
```

Метода `knn.NewKnnClassifier()` возвращает новый классификатор. Последний параметтр функции это количество соседей которые классификатор будет иметь.

Последняя часть `classify.go` показана ниже:

```
train, test := base.InstancesTrainTestSplit(rawData, 0.50)
cls.Fit(train)

p, err := cls.Predict(test)
 if err != nil {
 fmt.Println(err)
 return
}

confusionMat, err := evaluation.GetConfusionMatrix(test, p)

if err != nil {
 fmt.Println(err)
 return
}

fmt.Println(evaluation.GetSummary(confusionMat))
```

Работа с tensorflow

TensorFlow известная открытая платформа для ML. Для того чтобы использовать TensorFlow в Golang, нам нужно для начала скачать этот пакет.

```
 $ go get github.com/tensorflow/tensorflow/tensorflow/go
```

Однако, для работы вышеупомянутой команды, интерфейс C в TensorFlow должны быть уже установленны. На macOS машине, его можно установить таким образом:

```
$ brew install tensorflow
```

Если C интерфейс не установлен, и вы хотите установить Go пакет для TensorFlow, вы получите следующую ошибку:

```
$ go get github.com/tensorflow/tensorflow/tensorflow/go
# github.com/tensorflow/tensorflow/tensorflow/go
ld: library not found for -ltensorflow clang: error: linker command failed with exit code 1
(use -v to see invocation)
```

Так как TensorFlow достаточно сложен, он может быть хорош для выполнения следующей команды для проверки установки:

```
$ go test github.com/tensorflow/tensorflow/tensorflow/go

ok github.com/tensorflow/tensorflow/tensorflow/go  0.109s
```

Теперь начнем с некоторого кода:

```
package main

import (
  tf "github.com/tensorflow/tensorflow/tensorflow/go"
  "github.com/tensorflow/tensorflow/tensorflow/go/op"
  "fmt"
 )



func main() {
 s := op.NewScope()
 c := op.Const(s, "Using TensorFlow version: " + tf.Version())
```

```
 graph, err := s.Finalize()
 if err != nil {
  fmt.Println(err)
  return
 }

sess, err := tf.NewSession(graph, nil)

 if err != nil {
  fmt.Println(err)
  return
 }

output, err := sess.Run(nil, []tf.Output{c}, nil)
 if err != nil {
  fmt.Println(err)
  return
 }
 fmt.Println(output[0].Value())
}
```

Revision #3

Created 2021-12-26 08:57:23 UTC by gasick

Updated 2023-04-16 19:36:18 UTC by gasick