Universidad del País Vasco

Euskal Herriko Unibertsitatea

Técnicas de Inteligencia Artificial

Ingeniería Informática de Gestión y Sistemas de Información

Practica 3
Clasificacion

Autor(es):

Xabier Gabiña
Diego Montoya

6 de noviembre de 2024

Índice general

Índice de figuras

Índice de cuadros

Índice de Códigos

1. Introducción

Esta practica consiste en comprender e implmentar un clasificador mediante el algoritmo de perceptron. Para ello empezaremos con un clasificador simple usando puertas lógicas y posteriormente con el proyecto de un clasificador de dígitos y de caras para terminar con el proyecto del pacman.

2. Ejercicios

2.1. Perceptron

2.1.1. Descripción

2.1.2. Primera Implementación

```

```

Código 2.1: Implementación inicial del perceptron

2.1.3. Segunda Implementación

```python
class PerceptronClassifier:
  """
  Perceptron classifier.

  Note that the variable 'datum' in this code refers to a counter of features
  (not to a raw samples.Datum).
  """

  def __init__(self, legalLabels, max_iterations):
      self.legalLabels = legalLabels
      self.type = "perceptron"
      self.max_iterations = max_iterations
      self.weights = {}
      self.features = None
      for label in legalLabels:
          self.weights[label] = util.Counter()  # this is the data-structure you should use

  def setWeights(self, weights):
      assert len(weights) == len(self.legalLabels)
      self.weights = weights

  def train(self, trainingData, trainingLabels, validationData, validationLabels):
      """
      The training loop for the perceptron passes through the training data several
      times and updates the weight vector for each label based on classification errors.
      See the project description for details.

      Use the provided self.weights[label] data structure so that
      the classify method works correctly. Also, recall that a
      datum is a counter from features to values for those features
      (and thus represents a vector a values).
      """

      self.features = trainingData[0].keys()  # could be useful later
      # DO NOT ZERO OUT YOUR WEIGHTS BEFORE STARTING TRAINING, OR
      # THE AUTOGRADER WILL LIKELY DEDUCT POINTS.

      for iteration in range(self.max_iterations):
          print("Starting iteration ", iteration, "...")
          for i in range(len(trainingData)):  # training data
              # Obtener el ejemplo y su etiqueta real
              x_i = trainingData[i]
              y_i = trainingLabels[i]

              # Calcular los puntajes para cada etiqueta
              scores = util.Counter()
              for label in self.legalLabels:
                  scores[label] = self.weights[label] * x_i

              # Predecir la etiqueta con el puntaje más alto
```

```
51              predicted_label = scores.argMax()
52
53              # Si la predicción es incorrecta, actualizar los pesos
54              if predicted_label != y_i:
55                  self.weights[y_i] += x_i
56                  self.weights[predicted_label] -= x_i
57
58  def classify(self, data):
59      """
60      Classifies each datum as the label that most closely matches the prototype vector
61      for that label.  See the project description for details.
62
63      Recall that a datum is a util.counter...
64      """
65      guesses = []
66      for datum in data:
67          vectors = util.Counter()
68          for label in self.legalLabels:
69              vectors[label] = self.weights[label] * datum
70          guesses.append(vectors.argMax())
71      return guesses
72
73  def findHighWeightFeatures(self, label):
74      """
75      Returns a list of the 100 features with the greatest weight for some label
76      """
77      # Obtener los pesos de los features para la etiqueta dada
78      weights = self.weights[label]
79
80      # Ordenar los features por peso en orden descendente
81      sorted_features = weights.sortedKeys()
82
83      # Seleccionar los 100 features con mayor peso
84      featuresWeights = sorted_features[:100]
85
86      return featuresWeights
87
88
```

Código 2.2: Implementación final del perceptron

2.1.4. Casos de prueba

2.2. Clonando el Comportamiento del Pacman

2.3. Clonando el Comportamientodel Pacman con rasgos diseñados por nosotros

3. Resultados

3.1. Casos de prueba

3.1.1. Perceptron

```
python dataClassifier.py -c perceptron
Doing classification
--------------------
data:           digits
classifier:             perceptron
using enhanced features?:       False
training set size:      100
Extracting features...
Training...
Starting iteration  0 ...
Starting iteration  1 ...
Starting iteration  2 ...
Validating...
55 correct out of 100 (55.0%).
Testing...
48 correct out of 100 (48.0%).

```

Código 3.1: Ejecución del perceptron

3.1.2. Clonando el Comportamiento del Pacman

3.1.3. Clonando el Comportamiento del Pacman con rasgos diseñados por nosotros

3.2. Autograder

```


```

Código 3.2: Ejecución del autograder