Instance-based learning, *k*-nearest neighbor algorithm

Relevant Readings: Sections 8.1 and 8.2 in Mitchell

CS495 - Machine Learning, Fall 2009

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data
- Instance-based learning methods are “lazy” in the sense they delay the learning step

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data
- Instance-based learning methods are "lazy" in the sense they delay the learning step
  - "Training" consists of just storing the training data

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data
- Instance-based learning methods are "lazy" in the sense they delay the learning step
  - "Training" consists of just storing the training data
  - To classify a new instance, the training data is consulted directly

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data
- Instance-based learning methods are "lazy" in the sense they delay the learning step
  - "Training" consists of just storing the training data
  - To classify a new instance, the training data is consulted directly
- The good: These methods are straightforward, have competitive performance, and can usually be adapted to predicting continuous target functions

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data
- Instance-based learning methods are "lazy" in the sense they delay the learning step
  - "Training" consists of just storing the training data
  - To classify a new instance, the training data is consulted directly
- The good: These methods are straightforward, have competitive performance, and can usually be adapted to predicting continuous target functions
- The bad: The computational overhead can be large

Instance-based learning

- So far (checkers example, naive bayes) we have explicitly created a hypothesis based on training data
- Instance-based learning methods are "lazy" in the sense they delay the learning step
  - "Training" consists of just storing the training data
  - To classify a new instance, the training data is consulted directly
- The good: These methods are straightforward, have competitive performance, and can usually be adapted to predicting continuous target functions
- The bad: The computational overhead can be large

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled $+$ or -
- To classify a new instance, we simply return the label ($+$ or -) of the closest point from the training data

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -
- To classify a new instance, we simply return the label (+ or -) of the closest point from the training data
    - We can use the Euclidean distance to figure out which training point is closest

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -
- To classify a new instance, we simply return the label (+ or -) of the closest point from the training data
  - We can use the Euclidean distance to figure out which training point is closest
  - Distance from (x_1, y_1) to (x_2, y_2):

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -
- To classify a new instance, we simply return the label (+ or -) of the closest point from the training data
  - We can use the Euclidean distance to figure out which training point is closest
  - Distance from (x_1, y_1) to (x_2, y_2):
  - $dist((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -
- To classify a new instance, we simply return the label (+ or -) of the closest point from the training data
  - We can use the Euclidean distance to figure out which training point is closest
  - Distance from (x_1, y_1) to (x_2, y_2):
  - $dist((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- Question: What does the hypothesis space look like?

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -
- To classify a new instance, we simply return the label (+ or -) of the closest point from the training data
  - We can use the Euclidean distance to figure out which training point is closest
  - Distance from (x_1, y_1) to (x_2, y_2):
  - $dist((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- Question: What does the hypothesis space look like?
- Answer: A Voronoi diagram

Nearest neighbor algorithm (for 2 dimensionsional feature space)

- Suppose our data set consists of 2 continuous features
- Imagine each instance as a point in 2-dimensional *feature space*, labeled + or -
- To classify a new instance, we simply return the label (+ or -) of the closest point from the training data
  - We can use the Euclidean distance to figure out which training point is closest
  - Distance from (x_1, y_1) to (x_2, y_2):
  - $dist((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- Question: What does the hypothesis space look like?
- Answer: A Voronoi diagram

Nearest neighbor algorithm (for n dimensionsional feature space)

- What if we have n-dimensinal feature space?

Nearest neighbor algorithm (for n dimensionsional feature space)

- What if we have n-dimensinal feature space?
- We just use a more general notion of Euclidean distance

Nearest neighbor algorithm (for n dimensionsional feature space)

- What if we have n-dimensinal feature space?
- We just use a more general notion of Euclidean distance
  - Distance from $a = (a_1, a_2, \cdots, a_n)$ to $b = (b_1, b_2, \cdots, b_n)$:

Nearest neighbor algorithm (for n dimensionsional feature space)

- What if we have n-dimensinal feature space?
- We just use a more general notion of Euclidean distance
  - Distance from $a = (a_1, a_2, \cdots, a_n)$ to $b = (b_1, b_2, \cdots, b_n)$:
  - $dist(a, b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$

Nearest neighbor algorithm (for n dimensionsional feature space)

- What if we have n-dimensinal feature space?
- We just use a more general notion of Euclidean distance
  - Distance from $a = (a_1, a_2, \cdots, a_n)$ to $b = (b_1, b_2, \cdots, b_n)$:
  - $dist(a, b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$

What about noisy training data?

- If we simply use the closest training example, our classifier is sensitive to noisy data

What about noisy training data?

- If we simply use the closest training example, our classifier is sensitive to noisy data
- To combat this, we can take a majority vote of the k closest neighbors (where k is some integer constant such as 11)

What about noisy training data?

- If we simply use the closest training example, our classifier is sensitive to noisy data
- To combat this, we can take a majority vote of the k closest neighbors (where k is some integer constant such as 11)
  - Note that we should choose k to be odd so we don't get a tie

What about noisy training data?

- If we simply use the closest training example, our classifier is sensitive to noisy data
- To combat this, we can take a majority vote of the k closest neighbors (where k is some integer constant such as 11)
  - Note that we should choose k to be odd so we don't get a tie
- This is known as the k-nearest neighbor algorithm

What about noisy training data?

- If we simply use the closest training example, our classifier is sensitive to noisy data
- To combat this, we can take a majority vote of the k closest neighbors (where k is some integer constant such as 11)
  - Note that we should choose k to be odd so we don't get a tie
- This is known as the k-nearest neighbor algorithm
- Now if there is, say, a single incorrectly labeled training instance, it might not even effect the output at all (depending on the surrounding points in feature space)

What about noisy training data?

- If we simply use the closest training example, our classifier is sensitive to noisy data
- To combat this, we can take a majority vote of the k closest neighbors (where k is some integer constant such as 11)
  - Note that we should choose k to be odd so we don't get a tie
- This is known as the k-nearest neighbor algorithm
- Now if there is, say, a single incorrectly labeled training instance, it might not even effect the output at all (depending on the surrounding points in feature space)

How to choose k

- What is the "best" possible choice for k?

How to choose k

- What is the "best" possible choice for k?
  - It might depend on the dataset

How to choose k

- What is the "best" possible choice for k?
  - It might depend on the dataset
- Therefore, it would be sensible to have our algorithm decide how to select k

How to choose k

- What is the "best" possible choice for k?
  - It might depend on the dataset
- Therefore, it would be sensible to have our algorithm decide how to select k
- Question: But how do we do that?

How to choose k

- What is the "best" possible choice for k?
  - It might depend on the dataset
- Therefore, it would be sensible to have our algorithm decide how to select k
- Question: But how do we do that?
- Answer: Use a *tuning set*

How to choose k

- What is the "best" possible choice for k?
  - It might depend on the dataset
- Therefore, it would be sensible to have our algorithm decide how to select k
- Question: But how do we do that?
- Answer: Use a *tuning set*

Tuning sets

- The idea is to use some "make pretend testing data" to figure out what paramater (k, in the case of k-nearest neighbors) works best

Tuning sets

- The idea is to use some "make pretend testing data" to figure out what paramater (k, in the case of k-nearest neighbors) works best
- Simply to partition the training data into 2 parts: the training set and the *tuning set* (the faux testing set)

Tuning sets

- The idea is to use some "make pretend testing data" to figure out what paramater (k, in the case of k-nearest neighbors) works best
- Simply to partition the training data into 2 parts: the training set and the *tuning set* (the faux testing set)
  - Choose k so as to optimize performance on the tuning set

Tuning sets

- The idea is to use some "make pretend testing data" to figure out what paramater (k, in the case of k-nearest neighbors) works best
- Simply to partition the training data into 2 parts: the training set and the *tuning set* (the faux testing set)
  - Choose k so as to optimize performance on the tuning set
- You may also want to do cross-validation to decrease the risk of overfitting

Tuning sets

- The idea is to use some "make pretend testing data" to figure out what paramater (k, in the case of k-nearest neighbors) works best
- Simply to partition the training data into 2 parts: the training set and the *tuning set* (the faux testing set)
  - Choose k so as to optimize performance on the tuning set
- You may also want to do cross-validation to decrease the risk of overfitting

What about discrete attributes?

- How do we deal with discrete features?

What about discrete attributes?

- How do we deal with discrete features?
  - One way is to use a distance function where the distance between two points in that dimension is 1 unless the two feature values match exactly

What about discrete attributes?

- How do we deal with discrete features?
  - One way is to use a distance function where the distance between two points in that dimension is 1 unless the two feature values match exactly
  - This would need to be worked into the distance formula discussed earlier

What about discrete attributes?

- How do we deal with discrete features?
  - One way is to use a distance function where the distance between two points in that dimension is 1 unless the two feature values match exactly
  - This would need to be worked into the distance formula discussed earlier

Kernel functions

- Idea: how about we weight the closer training instances more heavily than ones that are far away?

Kernel functions

- Idea: how about we weight the closer training instances more heavily than ones that are far away?
- For example, the “weight” of a point could be taken as $1/d^2$ where d is the distance to that point

Kernel functions

- Idea: how about we weight the closer training instances more heavily than ones that are far away?
- For example, the "weight" of a point could be taken as $1/d^2$ where d is the distance to that point
- This is called a *kernel function*

Kernel functions

- Idea: how about we weight the closer training instances more heavily than ones that are far away?
- For example, the "weight" of a point could be taken as $1/d^2$ where d is the distance to that point
- This is called a *kernel function*
- If you do this, you don't need to restrict to the k nearest instances since far away points automatically have little effect on the outcome

Kernel functions

- Idea: how about we weight the closer training instances more heavily than ones that are far away?
- For example, the "weight" of a point could be taken as $1/d^2$ where d is the distance to that point
- This is called a *kernel function*
- If you do this, you don't need to restrict to the k nearest instances since far away points automatically have little effect on the outcome