Data Science

Introduction to Machine Learning:
Random Forests, k-NN, Feature Engineering

July 9, 2021

Goals for today

A quick survey of other ML methods

Goals for today

A quick survey of other ML methods

1. We've looked at two very different ways to build predictive models:

Goals for today

A quick survey of other ML methods

1. We've looked at two very different ways to build predictive models:
   1.1 Linear Regressions

Goals for today

A quick survey of other ML methods

1. We've looked at two very different ways to build predictive models:
    1.1 Linear Regressions
    1.2 Decision Trees

Goals for today

A quick survey of other ML methods

1. We've looked at two very different ways to build predictive models:
   1.1 Linear Regressions
   1.2 Decision Trees
2. What else is out there?

Previously, on...

Decisions Tress are great, but...

Previously, on...

Decisions Tress are great, but...

1. They are very brittle (prone to over-fit)

Previously, on...

Decisions Tress are great, but...

1. They are very brittle (prone to over-fit)
2. We discussed one way of combating that (Holdout Cross Validation)

Previously, on...

Decisions Tress are great, but...

1. They are very brittle (prone to over-fit)
2. We discussed one way of combating that (Holdout Cross Validation)
3. Is there any other way, specific to decision trees?

Previously, on...

Decisions Tress are great, but...

1. They are very brittle (prone to over-fit)
2. We discussed one way of combating that (Holdout Cross Validation)
3. Is there any other way, specific to decision trees?
4. Yes, Just make more trees!

Random forests

If one DT is brittle, make a bunch of trees to do the work together.

Random forests

If one DT is brittle, make a bunch of trees to do the work together.

1. Problem: How do we make sure that they're not all brittle in the same way?

Random forests

If one DT is brittle, make a bunch of trees to do the work together.

1. Problem: How do we make sure that they're not all brittle in the same way?
2. Solution (part 1): You don't use the whole dataset to train all of them.

Random forests

If one DT is brittle, make a bunch of trees to do the work together.

1. Problem: How do we make sure that they're not all brittle in the same way?
2. Solution (part 1): You don't use the whole dataset to train all of them.
3. Solution (part 2): You don't use *all* of the features in the DTs

Random forests

If one DT is brittle, make a bunch of trees to do the work together.

1. Problem: How do we make sure that they're not all brittle in the same way?
2. Solution (part 1): You don't use the whole dataset to train all of them.
3. Solution (part 2): You don't use *all* of the features in the DTs
4. These parts are known as *baggin* and *random feature selection*

That's my bag

Bagging is short for Bootstrap Aggregation:

That's my bag

Bagging is short for Bootstrap Aggregation:

1. From you dataset, sample (with replacement!) n elements

That's my bag

Bagging is short for Bootstrap Aggregation:

1. From you dataset, sample (with replacement!) n elements
2. For some number, B, of sample sets, train a DT for each $i \in B$

That's my bag

Bagging is short for Bootstrap Aggregation:

1. From you dataset, sample (with replacement!) n elements
2. For some number, B, of sample sets, train a DT for each $i \in B$
3. Now we need to Aggregate:

That's my bag

Bagging is short for Bootstrap Aggregation:

1. From you dataset, sample (with replacement!) n elements
2. For some number, B, of sample sets, train a DT for each $i \in B$
3. Now we need to Aggregate:
4. Thoughts on how?

That's my bag

Bagging is short for Bootstrap Aggregation:

1. From you dataset, sample (with replacement!) n elements
2. For some number, B, of sample sets, train a DT for each $i \in B$
3. Now we need to Aggregate:
4. Thoughts on how?
   4.1 It depends on whether you need a classifier!

Aggregate 1: You don't need a classifier

For each T_i we get some prediction y_i.

Aggregate 1: You don't need a classifier

For each T_i we get some prediction y_i.

1. For a regression, just take the average:

Aggregate 1: You don't need a classifier

For each T_i we get some prediction y_i.

1. For a regression, just take the average:
2. $\frac{1}{|B|} \sum_{j \in B} y_j$

Aggregate 2: You do need a classifier

For each T_i we get some prediction y_i.

Aggregate 2: You do need a classifier

For each T_i we get some prediction y_i.

1. For a classifier, just treat it as a vote!

RAS

Bagging ads some randomness, let's add more[1]

[1] it is a *random* forest, after all

RAS

Bagging ads some randomness, let's add more[1]

1. For each tree, at every split point (where you'll put a decision node):

[1] it is a *random* forest, after all

RAS

Bagging ads some randomness, let's add more[1]

1. For each tree, at every split point (where you'll put a decision node):
   1.1 choose a *random* subset of the available attributes (features)

[1] it is a *random* forest, after all

RAS

Bagging ads some randomness, let's add more[1]

1. For each tree, at every split point (where you'll put a decision node):
   1.1 choose a *random* subset of the available attributes (features)
   1.2 Use the same "most significant" attribute process as before, but only on that random subset.

[1] it is a *random* forest, after all

RAS

Bagging ads some randomness, let's add more[1]

1. For each tree, at every split point (where you'll put a decision node):
   1.1 choose a *random* subset of the available attributes (features)
   1.2 Use the same "most significant" attribute process as before, but only on that random subset.
2. This reduces the correlation *between* trees!

[1] it is a *random* forest, after all

Random Trees (classifier) with SKLearn:

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=N)
classifier = classifier.fit(X, Y)
```

Random Trees (regressor) with SKLearn:

```
from sklearn.ensemble import RandomForestRegressor

classifier = RandomForestRegressor(n_estimators=N)
classifier = classifier.fit(X, Y)
```

Other useful parameters:

Both use bootstraping by default (though you can turn it off)

Other useful parameters:

Both use bootstraping by default (though you can turn it off)

1. `max_depth`: How deep of a tree you'll allow

Other useful parameters:

Both use bootstraping by default (though you can turn it off)

1. `max_depth`: How deep of a tree you'll allow
2. `max_samples`: How many samples you'll allow for each T

Other useful parameters:

Both use bootstraping by default (though you can turn it off)

1. `max_depth`: How deep of a tree you'll allow
2. `max_samples`: How many samples you'll allow for each T
3. `min_samples_leaf`: How many samples needed for a leaf node to be created.

The company you keep

What if we don't want to think too hard about what our features mean?[2]

[2]This is a caricature!

The company you keep

What if we don't want to think too hard about what our features mean?[2]

1. The human brain is very good at pattern matching (this is good and bad)

[2] This is a caricature!

The company you keep

What if we don't want to think too hard about what our features mean?[2]

1. The human brain is very good at pattern matching (this is good and bad)
2. One way we determine what 'group' something belongs to is to see what it's near!

[2] This is a caricature!

The company you keep

What if we don't want to think too hard about what our features mean?[2]

1. The human brain is very good at pattern matching (this is good and bad)
2. One way we determine what 'group' something belongs to is to see what it's near!
3. Let's write a program to do that.

[2] This is a caricature!

k-NN

First let's define a *neighbor*

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:
   2.1 Is B or C closer to point A?

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:
   2.1 Is B or C closer to point A?
3. We can ask this for every point in the dataset, for *every other* point in the dataset

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:
   2.1 Is B or C closer to point A?
3. We can ask this for every point in the dataset, for *every other* point in the dataset
4. This can give us an *ordering* of the nearest neighbors.

$k\ldots$

k...

1. When $k = 1$, we only care about the closet point.

k...

1. When $k = 1$, we only care about the closet point.
2. When $k = 2$, we only care about the two closet points, and they ‘vote’.

k...

1. When $k = 1$, we only care about the closet point.
2. When $k = 2$, we only care about the two closet points, and they ‘vote’.
3. When $k = 3$, we only care about the... you get it.

Discussion time.

What do we have to do in order to train this model?

Discussion time.

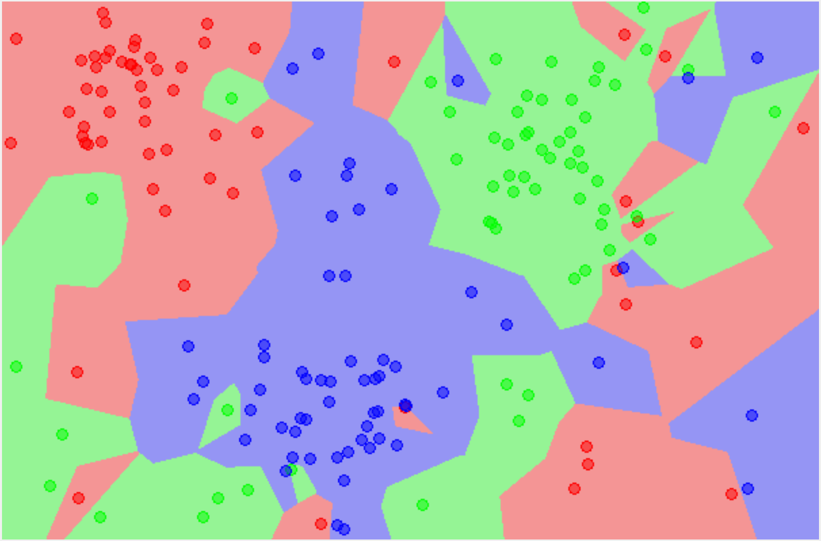
What do we have to do in order to train this model?

1. We only have to store the dataset in a way that let's us calculate the nearest neighbor(s)

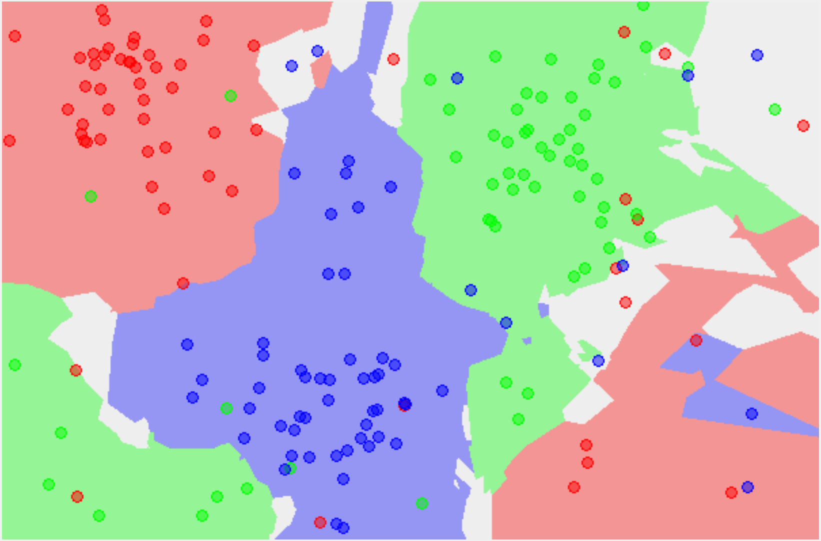
Discussion time 2.

What do we think k-NN might be good for?

Let's take a look (1NN):



Let's take a look (5NN):



k-NN and SKLearn

You might see a pattern developing:

k-NN and SKLearn

You might see a pattern developing:

1. `sklearn.neighbors.KNeighborsClassifier`

k-NN and SKLearn

You might see a pattern developing:

1. `sklearn.neighbors.KNeighborsClassifier`
2. Defaults to $k = 5$

k-NN and SKLearn

You might see a pattern developing:

1. `sklearn.neighbors.KNeighborsClassifier`
2. Defaults to $k = 5$
3. You can also play with the `metric` and `p` parameters to change how distance is calculated (default is Euclidean distance).

Feature Engineering (interaction terms)

Feature Engineering (interaction terms)

1. Sometimes (as in the video for the make up lecture) we only want to add categorical terms

Feature Engineering (interaction terms)

1. Sometimes (as in the video for the make up lecture) we only want to add categorical terms
2. But sometimes we also want to add *interactions* between our terms.

Feature Engineering (interaction terms)

1. Sometimes (as in the video for the make up lecture) we only want to add categorical terms
2. But sometimes we also want to add *interactions* between our terms.
3. To the notebook!

Thanks for your time!

:)