An Application of Markov Model

In this lesson, let's have a look at an application of Markov Model: Randomized Text Generation.

WE'LL COVER THE FOLLOWING

In the previous lesson, we implemented the *Markov process distribution*, which is a distribution over state sequences, where the initial state and each subsequent state is random. There are lots of applications of Markov processes. In this lesson, we will have a look at **randomized text generation**.

Randomized Text Generation

Randomized text generation using a Markov process has a long history on the internet; one of our favorite examples was “**Mark V. Shaney**”.

> Mark V. Shaney is a synthetic Usenet user whose postings in the *net.singles* newsgroups were generated by Markov chain techniques, based on text from other postings. The username is a play on the words “Markov chain”. We can now think of it as a “bot”, that was running a Markov process generated by analysis of USENET posts, sampling from the resulting distribution to produce a “Markov chain” sequence and posting the resulting generated text right back to USENET.

In this lesson, we are going to replicate that. The basic technique is straightforward:

1. Start with a corpus of texts.
2. Break up the text into words.
3. Group the words into sentences.
4. Take the first word of each sentence and make a weighted distribution; the more times this word appears as a first word in a sentence, the higher it is weighted. This gives us our initial distribution.
5. Take every word in every sentence. For each word, generate a distribution of the words which follow it in the corpus of sentences. For example, if we have “frog” in the corpus, then we make a distribution based on the words which follow: “prince” twice, “pond” ten times, and so on. This gives us the transition function.
6. There are several ways to handle words which end sentences, but an easy way to do it is to have a bunch of “end words” where the transition function gives the empty distribution.
7. Sample repeatedly, and format the sequences of words back into sentences.
8. Hilarity!

Let’s do it.

All our probability distributions thus far have been immutable, and we should continue that tradition. We’ll use the builder pattern to construct immutable objects. We’ll start by making a **categorical distribution builder**, and then make a Markov builder from that:

```
sealed class DistributionBuilder<T>
{
  private Dictionary<T, int> weights = new Dictionary<T, int>();
  public void Add(T t)
  {
    weights[t] = weights.GetValueOrDefault(t) + 1;
  }
  public IDistribution<T> ToDistribution()
  {
    var keys = weights.Keys.ToList();
    var values = keys.Select(k => weights[k]);
    return keys.ToWeighted(values);
  }
}
```

> Recall that we do not need to worry that we've closed over a member that is a mutable collection here because `ToWeighted` is going to call `ToList` on the values.

Our Markov process builder is only slightly more complicated:

```
sealed class MarkovBuilder<T>
{
  private DistributionBuilder<T> initial =
    new DistributionBuilder<T>();
  private Dictionary<T, DistributionBuilder<T>> transitions =
    new Dictionary<T, DistributionBuilder<T>>();
  public void AddInitial(T t)
  {
    initial.Add(t);
  }
  public void AddTransition(T t1, T t2)
  {
    if (!transitions.ContainsKey(t1))
      transitions[t1] = new DistributionBuilder<T>();
    transitions[t1].Add(t2);
  }
  public Markov<T> ToDistribution()
  {
    var init = initial.ToDistribution();
    var trans = transitions.ToDictionary(
      kv => kv.Key,
      kv => kv.Value.ToDistribution());
    return Markov<T>.Distribution(
      init,
      k => trans.GetValueOrDefault(k, Empty<T>.Distribution));
  }
}
```

Generating Text from Shakespeare Corpus

Super! Now, all we need is the complete works of Shakespeare, thank you Project Gutenberg. We don't need to load the whole five megs into memory at once; we can process it line-at-a-time thanks to our builder.

Word Splitting

Word Splitting

We’ll start by splitting up the line stream into a word stream:

```
static readonly char[] punct = "<>,*-()[#]@:%\"/';_&}".ToCharArray();
static IEnumerable<string> Words(
    this IEnumerable<string> lines) =>
  from line in lines
  from word in line.Split()
  let raw = word.Trim(punct)
  where raw != ""
  select raw;
```

This is a simple word-splitting algorithm. We are stripping out any leading or trailing punctuation, but we are leaving in the periods, exclamation marks and question marks. This means that we are stripping out the single trailing quotes in Shakespearean contractions like *th’*.

We are also not canonicalizing the casing; we’ll be treating “*The*” and “*the*” as different words, and so on. This is not necessarily the best approach.

> **Exercise:** Can you come up with a better word-splitting algorithm than this one? This will do for our lesson, but it would be nice to have a better algorithm.

Splitting into Subsequences

A generalized *" split a sequence into subsequences"* operator is a handy tool to have that is not included in the LINQ sequence operators; let’s make one:

```
public static IEnumerable<List<T>> Split<T>(
  this IEnumerable<T> items,
  Func<T, bool> last)
{
  var list = new List<T>();
  foreach (T item in items)
  {
    list.Add(item);
    if (last(item))
    {
      yield return list;
      list = new List<T>();
    }
  }
  if (list.Any())

    yield return list;
}
```

Sentence Extractor

And now making a sentence-extractor is straightforward:

```
static IEnumerable<List<string>> Sentences(
    this IEnumerable<string> words) =>
  words.Split(w => {
    char c = w[w.Length - 1];
    return c == '.' || c == '!' || c == '?';
  });
```

Again, not great; this would make "*Mr.*" the end of a sentence, for instance. Notice again that we are leaving the punctuation in the word. Perhaps not the best choice, but it will work for our purposes.

Put it all together:

```
var builder = new MarkovBuilder<string>();
var sentences = File.ReadLines("shakespeare.txt")
  .Words()
  .Sentences();
foreach (var sentence in sentences)
{
  builder.AddInitial(sentence[0]);
  for (int i = 0; i < sentence.Count - 1; i += 1)
    builder.AddTransition(sentence[i], sentence[i + 1]);
}
var markov = builder.ToDistribution();
```

Excellent, and now, release the Markovian monkey, and we'll see if *Hamlet* emerges:

```
Console.WriteLine(markov
  .Samples()
  .Take(100)
  .Select(x => x.SpaceSeparated())
  .NewlineSeparated());
```

Here's a slightly-cleaned-up-for-legibility sample output; I've moved around the line breaks to make it easier to read.

```
SCENE I.

POET.

A very echo with him there. I say away!

HECTOR.

You know the body. Sir by my love him
an end. But your king? Sir John of a purse
I grant that thou brought forth in them with ears.
What's the rebels with him. Well and others
orchards grew together.

BERTRAM.

Tis time his horse?

Dost thou seek how well but a shame this is but with
the Duke of Eve's legacy be the counter-gate which
bears will stand till he hath great world shall be a
woman with my ever is to cunning bade him
for recordation to death.

Exit GLOUCESTER

Re-enter Ghost.

CYMBELINE.

Grim-visag'd war but since thou turn the morn-dew
on this wretched soul elected him fair Helen
then let all our actors are you?

Now are both mine and said well his pitcher
by the heat of mine own desert?

What's o'clock? I may not you do
fight with you.
The mother breeds suspicion hath!
Be soon shot.
```

Be soon shot.

Exeunt ACT III.
Let’s leave it at that.

It’s not *Hamlet*, to be sure, but it’s a start.

If you try this on the complete works of *Jane Austen*, something like this will be produced.

The principal part of the result of your own line from any reply to the ecstasies of meeting he gravely but it in Hertfordshire and I mean to declare I have the days brought you must not. You must be happier than in Highbury with awful object as a greater steadiness had reached the picture of consequence to any objection to each other day came out.

Implementation

Let’s have a look at the complete code now:

Program.cs

Bernoulli.cs

BetterRandom.cs

Distribution.cs

DistributionBuilder.cs

Empty.cs

Episode25.cs

Extensions.cs

IDiscreteDistribution.cs

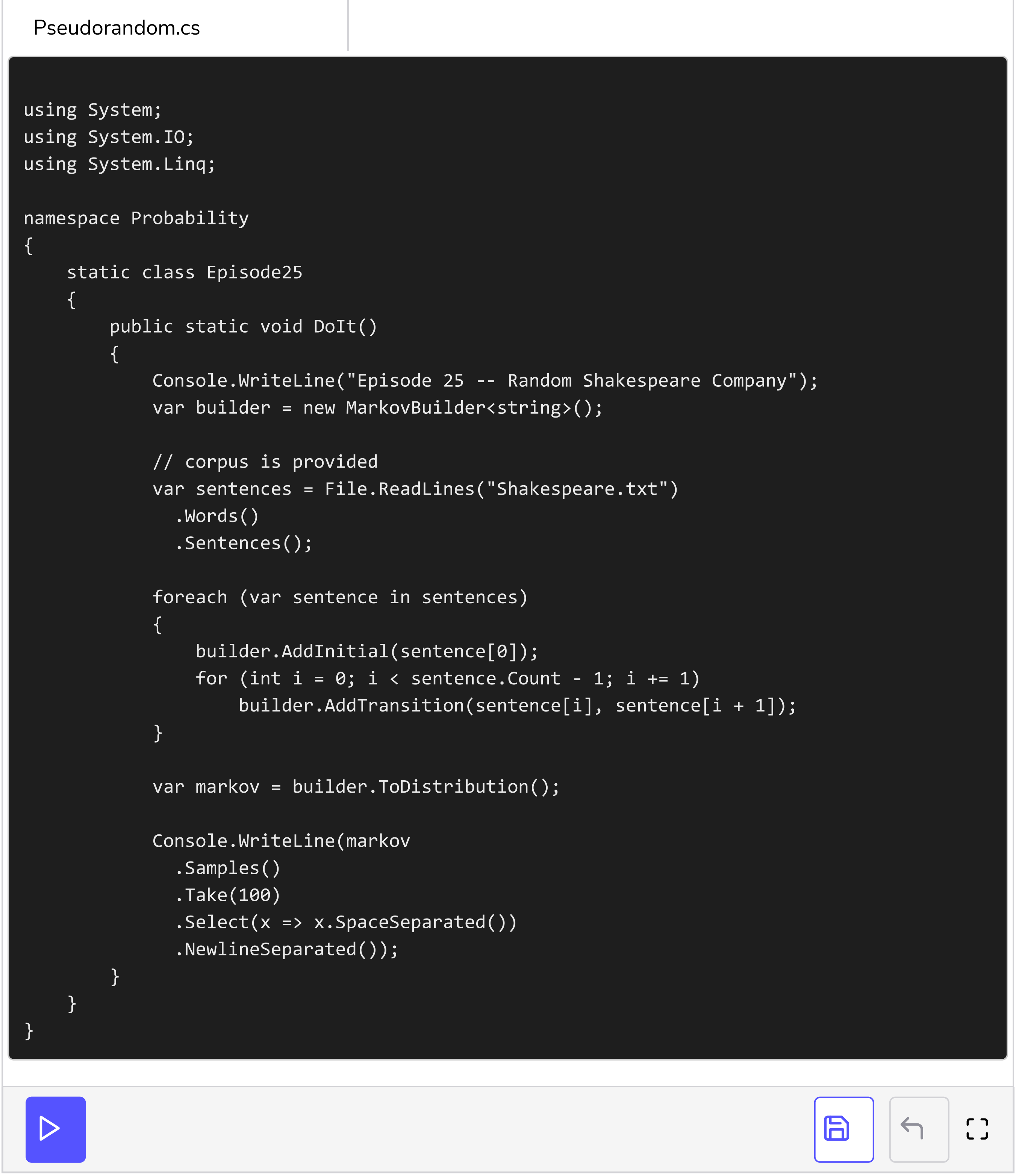
IDistribution.cs

Markov.cs

MarkovBuilder.cs

Normal.cs

Projected.cs



```
using System;
using System.IO;
using System.Linq;

namespace Probability
{
    static class Episode25
    {
        public static void DoIt()
        {
            Console.WriteLine("Episode 25 -- Random Shakespeare Company");
            var builder = new MarkovBuilder<string>();

            // corpus is provided
            var sentences = File.ReadLines("Shakespeare.txt")
              .Words()
              .Sentences();

            foreach (var sentence in sentences)
            {
                builder.AddInitial(sentence[0]);
                for (int i = 0; i < sentence.Count - 1; i += 1)
                    builder.AddTransition(sentence[i], sentence[i + 1]);
            }

            var markov = builder.ToDistribution();

            Console.WriteLine(markov
              .Samples()
              .Take(100)
              .Select(x => x.SpaceSeparated())
              .NewlineSeparated());
        }
    }
}
```

In the next lesson, we will turn our attention back to continuous distributions.