Exon Prediction Using HMM Algorithm

Table of Contents

1. Introduction

The goal of our program is to take a given gene transcript and use our program to predict the position and the number of exons the gene has through implementation of the Hidden Markov Model

(HMM). HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process with hidden states[2]. It provides a conceptual toolkit for building complex models just by drawing an intuitive picture. This model could be used for a diverse range of problems such as genefinding and multiple sequence alignment. We compared our algorithm with HMMgene and AUGUSTUS which are published tools from other universities.

AUGUSTUS is a program that predicts genes in eukaryotic genomic sequences[1]. It is a method in which genomic DNA is systematically searched for potential coding genes and prediction. It can report a large number of alternative genes which includes the possibilities for the transcripts and each exon and intron[1]. HMMgene mainly searches for exons and splice sites within a given gene[7]. We utilized these 2 published tools, in addition to NCBI Genbank, to compare our prediction results as well as the validity of other published tools.

2. Data sets

The data sets used in this project were collected from NCBI GenBank (NCBI) and UCSC Genome Browser (UCSC). For this project, we used 20 gene sequences from different species (Figure 1). For convenience, we renamed the sequences that we used in our programs, so that our programs can read them easily. We will use the abbreviation names and/or species common names in the report for the reader to understand our process and analysis without reading the complex names (Figure 1). Our criteria for selecting the genes were, each gene needs to have multiple exons and each gene is on a different strand. The first gene, SLC25A6, encodes for a pore in the mitochondrial membrane to transport ADP and ATP. The second gene, PRPS2, encodes for a synthetase that makes DNA bases. However, the gene product and protein function was not a focus of our program's purpose.

The first step in gathering datasets for our program started with using UCSC to identify a gene in humans that is shared with 9 other vertebrate species. Next, we obtained the genomic sequences from assembly through UCSC. Even though we could also obtain the same data directly from NCBI, UCSC allowed us to obtain the sequences with exons and introns formatted in uppercase and lowercase, respectively. Initially, we did not limit our data selections to only vertebrates. However, when we tested our datasets with the publish tools, we encountered some limitations. HMMgene can only generate predictions for humans, *C. elegans*, and other vertebrates while AUGUSTUS can generate predictions for many different species, but we had to limit them to chicken, human, and zebrafish because these were more commonly found through UCSC and NCBI. We then repeated this process for a second set of sequences.

We applied our own programs on 10 SLC25A6 sequences (Figure 1, top) while we applied the HMMgene tool on all of them and AUGUSTUS a selected few (Figure 1). The purpose of the second set of sequences is to verify the validity of the published tools which we used to compare with our program.

The first gene, SLC25A6, is encoded on the complementary strand while the second gene, PRPS2, is encoded on the forward strand. Therefore, having 2 genes on different strands will help us determine if our published tools can make accurate predictions since we are using those results to compare with our program's output.

SLC25A6	Species Name	Name used in report	Predict with own codes
NM_174660.2	Bos taurus_cow	seq1	yes
NM_001128732.1	Danio rerio_zebrafish	seq2	yes
NM_001303771.1	Esox lucius_pike	seq3	yes
NM_204231.2	Gallus gallus_chicken	seq4	yes
NM_001636.4	Homo sapiens_human	seq5	yes
NM_001201130.1	Ictalurus_punctatus_catfish	seq6	yes
NM_001284659.1	Macaca fascicularis_macaque	seq7	yes
NM_001127280.1	Ovis aries_sheep	seq8	yes
NM_214418.2	Sus scrofa_boar	seq9	yes
NM_001044374.1	Takifugu rubripes_puffer fish	seq10	yes

PRPS2	Species Name	Name used in report	Predict with own codes
NM_001114152.1	Bos taurus_cow	P1	no
NM_001006264.1	Gallus gallus_chicken	P2	no
NM_002765.4	Homo sapiens_human	P3	no
NM_001201074.1	Ictalurus punctatus_catfish	P4	no
NM_001319562.1	Macaca fascicularis_macaque	P5	no
NM_026662.5	Mus musculus_mouse	P6	no
NM_001280389.1	Pan troglodytes_chimpanzee	P7	no
NM_001135526.1	Pongo abelii_orangutan	P8	no
NM_012634.1	Rattus norvegicus_rat	P9	no
NM_001243801.1	Sus scrofa_boar	P10	no

Figure 1. Dataset sorted by species name and accession number. The selected sequences that were tested with AUGUSTUS are seq5, P2, and P3.

3. Methods

3.1 HMM

An hidden markov model is a machine learning technique that uses samples to train a model and then score another sequence with unknown states with forward algorithm, backward algorithm and baum-welch algorithm (or other algorithms, but I didn't use them in our code). For this technique, we need to define the observations which we can see and the hidden states which we cannot see and the HMM is going to predict. In this project, the observations are the nucleotides (A, T, C, G), therefore, it has 4 observations. The hidden states are exon and intron, so it has 2 states.

In the training part of HMM, it will finally generate a model with 3 matrices. A pi matrix is an initial state matrix that tells us the probability that how likely a sequence will start at each state. An A

matrix is called a transition matrix that tells us the probability between two states. A B matrix is a matrix of emission probabilities that describe the relations between each observation and each state. In the part of scoring, we will use the model we just trained and a sequence with unknown states to find out the most likely states sequence. That is, we want to find out the corresponding exon or intron for each nucleotide in the sequence that we use to score.

3.2 Unsupervised HMM

We will use unsupervised HMM with C++ and python. An unsupervised HMM is that we won't tell the machine the corresponding state of each observation in sequence when we train the model. Then, let it generate the model by analyzing the pattern of the sequence. In the python code, we import the HMM from HMMlearn.

3.3 Supervised HMM

While we do not label the data in an unsupervised technique training process, we do label observations sequence in a supervised HMM. Therefore, if the observation is uppercase, we will tell the machine and label it. In this part, we used the seqlearn which is the extension of the sklearn.

3.4 Cross Validation

In this project we applied a technique called cross validation to minimize the bias. We used the same data sets to train and test. For example, if we want to use seq1 and seq2 to predict the hidden states (the exon or intron), we will use seq3~seq10 as the training data. We repeated this similar step five times to gain the prediction of seq1~seq10.

3.5 Prediction of Hidden States Versus Scoring

When we try to predict the most likely intron or exon sequence for the test data, we use the backward algorithm to calculate the highest probability of the state in each nucleotide position. The state that has the highest probability will be the expected state in that position.

In the scoring part that we also did as some additional work, we try to separate the sequence of the exon and the sequence of intron. We only use the sequence of the exon to train a model, and see how other exon or intron sequences fit this model. We gain scores to classify them.

4. Organization

4.1 The roadmap of the C++ code

(1)read the training file and translate the letters of nucleotides into integer format; (2)save the “integer” nucleotide into a file; (3)read the file and input the integer sequence into a pointer; (4)create numbers of the training models and start to train; (5)in the training process, initialize a hmm model with 2 states and 4 observations; (6)initialize the pi matrix, A matrix, and B matrix with random numbers; (7)computer the alpha pass, beta pass, gammas, and digammas, then re-estimate the pi, A, B matrix; (8) after train a model based on the first training set, keep training on the other training sets without initialize the model; (10)

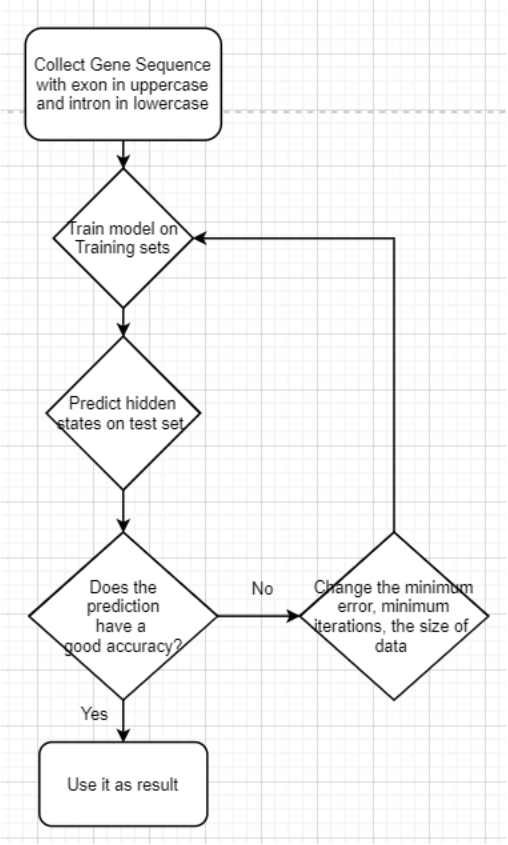
after many iteration, the code will choose the best model that has highest score and move to prediction process; (11) in the prediction process, read the test file and translate the nucleotide into number, then save in a file; (12)read from that file to get the sequence, and apply the model we just train on this sequence; (13)choose the most likely state for each nucleotide position.

4.2 The roadmap of the python unsupervised code

(1)this is using the hmmlearn library. It reads the lists of the test files, translate the letters of nucleotides into number; (2)create an array of the training sequences then use hmm gaussianHMM to train a model; (3)use the model we just train to do the prediction on the test files

4.3 The roadmap of the python supervised code

(1)this is using the seqlearn. It is very similar to the unsupervised code in hmmlearn. (2) the differences are: a feature that labels the sequence, and it has to be changed into CoNLL 2000 format.



5. Design

(1) Initialize a HMM model;
(2) Input training data;
(3) Train and gain the model with pi, A and B matrix;
(4) Input test data;
(5) Predict the hidden states of the observations in the sequence of test data;
(6) Compare the predictional states sequence with the original states sequence to find the accuracy.
(7) Align the predictional states sequence with the original sequence of observations.

Some Additional work:

(8) Use the sequence of genes from human species only to redo the process 1~7;
(9) Use the exon sequences only to train a HMM model, then use this model score on the intron and exon sequences to make classification.
(10) Use the results and data to generate graphs to visualize the results.

6. Sample output

6.1 Result of Training Model in C++

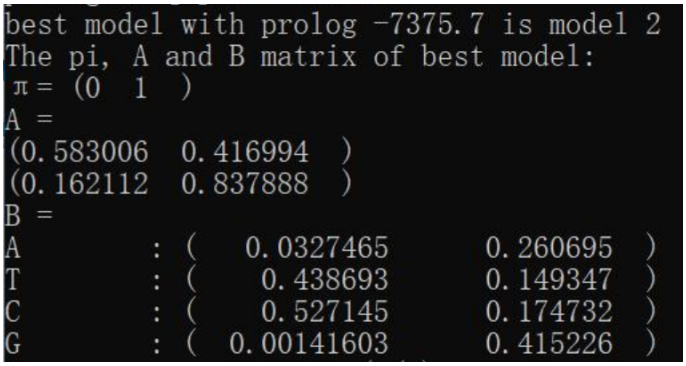


Figure2: A Model After Training

This picture shows us that the initial matrix (pi matrix), the transition matrix (A matrix) and the matrix of emission probabilities (B matrix). We can see each column represent one state, but we cannot determine which column is exon or intron. However, we can identify that adenine and guanine has much lower probabilities in the first column.

6.2 Result of Prediction in C++

	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7	Seq8	Seq9	Seq10
Correct Rate of Prediction(%)	60.82	61.19	60.19	60.07	57.61	60.43	57.45	60.08	61.5	61.69

Figure3: The Correct Rate of Prediction in C++ HMM

The table only shows a correct rate around 60%. For the seq5, and seq7, the numbers are even lower than 60%. The problem may be that we don't use large enough data to train the model. Another probable reason may be that the gene sequences come from different species.

```
C  C  T  G  C  C  A  C  C  A  T  G  A  C  G  G  A  A  C  A
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1

G  G  C  C  A  T  C  T  C  C  T  T  C  G  C  C  A  A  A  G
1  1  1  1  1  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1

A  C  T  T  C  T  T  G  G  C  C  G  G  A  G  G  C  A  T  C
1  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1

G  C  C  G  C  C  G  C  C  A  T  C  T  C  C  A  A  G  A  C
1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1

G  G  C  C  G  T  G  G  C  T  C  C  G  A  T  C  G  A  G  C
1  1  1  1  1  1  1  1  0  0  0  0  1  1  1  1  1  1  1  1

G  G  G  T  C  A  A  G  C  T  G  C  T  G  C  T  G  C  A  G
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

Figure4: Partial Alignment of The Sequence and Prediction in Unsupervised HMM

6.3 Result of Prediction in Python Unsupervised HMM (HMMlearn)

	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7	Seq8	Seq9	Seq10
Correct Rate of Prediction(%)	52.96	53.0	53.07	53.02	52.06	53.21	52.1	53.22	52.82	52.72

Figure5: The Correct Rate of Prediction in Python HMMLearn

The results that we gained from the python HMMlearn are not better than the one we gained from C++ program. Since HMM is an unsupervised technique, it will look for the patterns or rules from the training sets. Therefore, if it doesn't have large enough data, it will be hard to make a good prediction.

6.4 Result of Prediction in Python supervised HMM (seqlearn)

	Seq1	Seq2	Seq3	Seq4	Seq5	Seq6	Seq7	Seq8	Seq9	Seq10
Correct Rate of Prediction(%)	97.29	97.42	97.26	97.45	88.88	92.94	89.16	93.21	94.25	94.26

Figure6: The Correct Rate of Prediction in Python Seqlearn

```
C  C  T  G  C  C  A  C  C  A  T  G  A  C  G  G  A  A  C  A
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  1  1

G  G  C  C  A  T  C  T  C  C  T  T  C  G  C  C  A  A  A  G
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0

A  C  T  T  C  T  T  G  G  C  C  G  G  A  G  G  C  A  T  C
0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1

G  C  C  G  C  C  G  C  C  A  T  C  T  C  C  A  A  G  A  C
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  1

G  G  C  C  G  T  G  G  C  T  C  C  G  A  T  C  G  A  G  C
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1

G  G  G  T  C  A  A  G  C  T  G  C  T  G  C  T  G  C  A  G
1  1  1  1  1  0  0  1  1  1  1  1  1  1  1  1  1  1  0  0

G  T  G  g  g  g  a  c  g  c  g  g  g  c  g  c  g  g  c  c
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure7: Partial Alignment of The Sequence and Prediction in Supervised HMM

The prediction from the seqlearn code has a pretty high correction. However, it only based on the "labeled" observations sequence that we told it. If we use nucleotides with all lowercase or all uppercase, it won't work well.

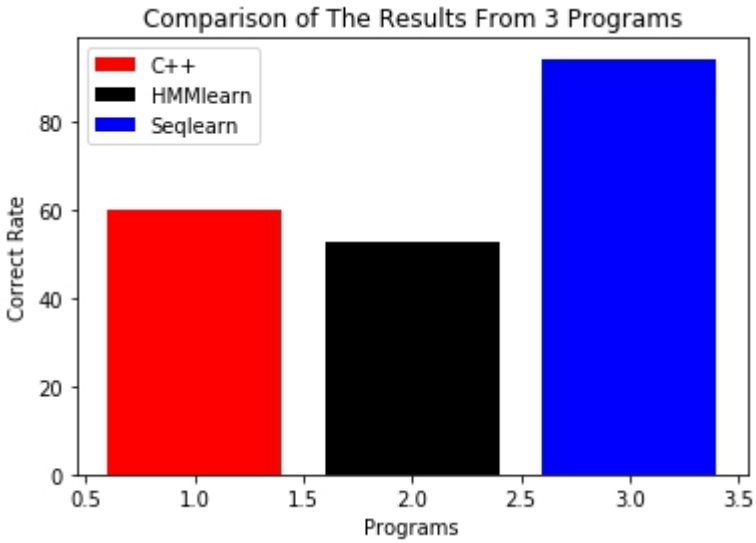


Figure8: Comparison of The Results From 3 Programs

6.5 Result of Classifying the Exon and Intron

We only used the small pieces of exon segments to train a model. Then, we used this model to score on the exon and intron separately and gain the scores as below:

	Exon1	Exon2	Exon3	Exon4	Exon5
Score	-1.4301	-1.36292	-1.38285	-1.38275	-1.41926

	Exon6	Exon7	Exon8	Exon9	Exon10
Score	-1.36678	-1.39873	-1.36005	-1.42745	-1.40008

	Intron1	Intron2	Intron3	Intron4	Intron5
Score	-1.35932	-1.390688	-1.38084	-1.42745	-1.375526

Figure9: Scores for The Exons and Introns

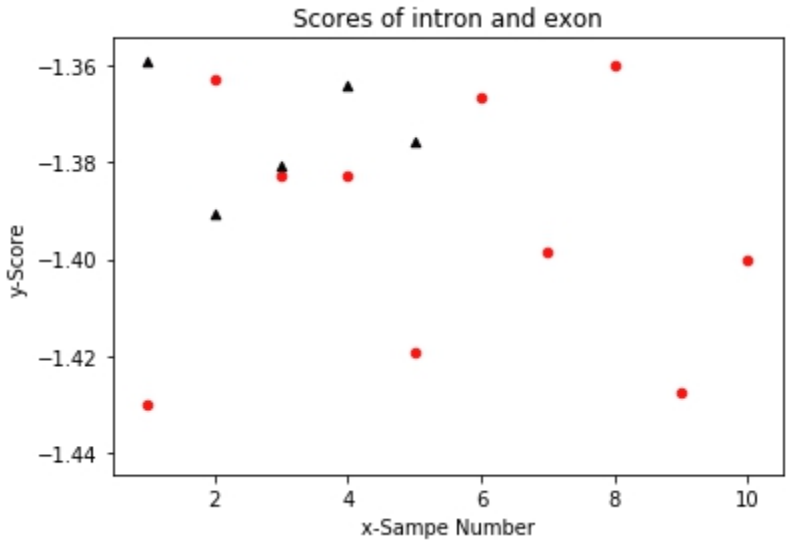


Figure10: Scatter Plot of The Exons and Introns

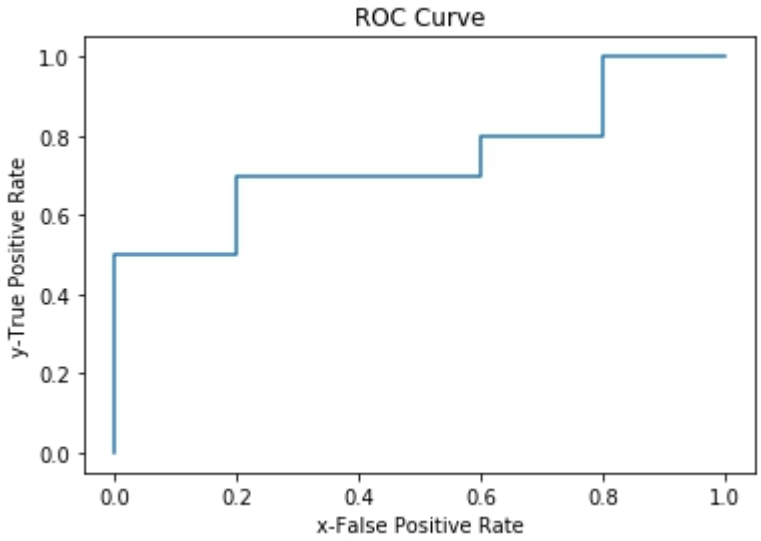


Figure11: ROC Curve

The reason why we made the graphs of Scatter Plot and ROC Curve is that we want to find a threshold that we can classify exons and introns based on their score from our training model. And with this threshold, we can have a higher true positive rate and relatively lower false positive rate. In the ROC Curve, we can identify that the threshold is around -1.38. That means if the score is lower than -1.38, it tends to find an exon while if the score is higher than -1.38, it tends to find an intron.

7. Execution of our program

For instruction of C++ code, you can put all the data sets of text format in the folder of the programs. Then, you need to put the training files name in the command argument (only write the names of the files that you want to train), for example, “seq1.txt”. Then, the programs will ask you “how many

training sets you use" and "how many training models you want". Since I am using pointers in this program, if there are too many training models, the programs will crash. Therefore, it's better to use less than 10 training models. After the training process, it will ask you if you want to score on any file. You will type "Y" as yes. Then, you can type in the file's name that you want to predict. Then, it will ask you the output file's number you want. If you want to predict for the seq1, you'd better write "1" as the output file' number, and it will create a file called "prediction1.txt". In the output file, it will use "1" and "0" to represent 2 states.

For the python code instruction, you just need to place the files under the same folder of the ".py" files. And then, you can replace the training files' and testing file's name in the codes, and also the output files' name. When it runs, it will not ask you to type anything.

8. Implication of published tools

The 2 published tools that we also implemented are HMMgene and AUGUSTUS, as mentioned above. Both tools make predictions using an existing reference genome which would be compared with our input FASTA file. For simplicity, we will only discuss the prediction results for seq5 and P3 on both HMMgene and AUGUSTUS.

For seq5, NCBI reported 4 exons with the positions at (105-215, 2387-2873, 4695-4835, and 5355-5512). HMMgene predicted all 4 exons correctly. AUGUSTUS, however, only predicted the first 2 exons correctly and it only predicted 3 exons total. Additionally, AUGUSTUS predicted that there are 2 possible genes on the same given sequence. Another notable detail is that HMMgene and AUGUSTUS stated that these predictions are for the forward strand. However, according to NCBI, this gene is on the complementary strand.

For P3, NCBI reported 7 exons with the positions at (144-265, 7853-8036, 17880-17978, 18668-18792, 28153-28326, 29290-29449, 31350-31442). HMMgene predicted only the first 3 exons correctly and it predicted a total of 8 exons. AUGUSTUS, however, predicted all 7 exons' positions correctly. This time, it predicted only 1 gene on the strand. Both HMMgene and AUGUSTUS predicted the gene for the forward strand in this case, which is consistent with NCBI.

These comparisons imply that different programs may result in different predictions depending on how it was trained and what kinds of datasets were used during that process. Therefore, our most reliable source to evaluate our own program would most likely be NCBI. However, it was useful to utilize other tools and understand that there are various ways to execute gene prediction and each way may give a unique result, which we would have to further analyze and only retain the most accurate output.

9. Conclusion

The results that we gained from C++ HMM code is better than the one we gained from using the library of the HMMlearn. The code used from seqlearn, the supervised HMM, has the highest predictions.

However, it's not very practical because it requires the user to label the sequence for training. The next steps we need to do are to improve the C++ and python code to make a higher correct rate of prediction, and make the seqlearn code to fit the general data. Instead of using only one nucleotide as each observation in the sequence, we can think about using a small segment of nucleotides as each observation. For example, we could observe 3 nucleotides at a time which means we will have 64 different combinations (observations). And we can predict the motif instead of each nucleotide. We believe that will have a higher accuracy.

10. References

1) "The AUGUSTUS Gene Prediction Tool." *Augustus: Gene Prediction*, bioinf.uni-greifswald.de/augustus/.
2) Eddy, Sean R. "What Is a Hidden Markov Model?" *Nature News*, Nature Publishing Group, www.nature.com/articles/nbt1004-1315.
3) "GenBank Overview." *National Center for Biotechnology Information*, U.S. National Library of Medicine, www.ncbi.nlm.nih.gov/genbank/.
4) Krogh, A : *Two methods for improving performance of an HMM and their application for gene finding.* In Proc. of Fifth Int. Conf. on Intelligent Systems for Molecular Biology, ed. Gaasterland, T. et al., Menlo Park, CA: AAAI Press, 1997, pp. 179-186.
5) "Revealing Genome Features." *Understanding Bioinformatics*, by Marketa J. Zvelebil and Jeremy O. Baum, Garland Science, 2008, pp. 317–356.
6) "Services." *Https://Www.healthtech.dtu.dk*, services.healthtech.dtu.dk/service.php?HMMgene-1.1.
7) "UCSC Genome Browser Home." *UCSC Genome Browser Home*, genome.ucsc.edu/.