AI for Genomics: from CNNs and LSTMs to Transformers

Nikolay Oskolkov, Group Leader (PI) at LIOS, Riga, Latvia

Physalia course, 09.09.2025

Session 3a: CNNs and RNNs applications for genomics and ancient genomics

@NikolayOskolkov

@oskolkov.bsky.social

Personal homepage:
https://nikolay-oskolkov.com

Image generated at www.canva.com

Session content

Topics we'll cover in this session:

1) Introduction to ancient genomics: contamination problem

2) Decontaminating ancient data with neural networks

3) Recognizing introgressed regions with neural networks

4) One-hot-encoding of biological nucleotide sequences

5) Training CNNs and RNNs on genomics data

Deep Learning for DNA sequencing

PERSPECTIVE
https://doi.org/10.1038/s41588-018-0295-5
nature genetics

A primer on deep learning in genomics

James Zou[1,2,3]*, Mikael Huss[4,5], Abubakar Abid[3], Pejman Mohammadi[6,7], Ali Torkamani[6,7] and Amalio Telenti[6,7]*

Deep learning methods are a class of machine learning techniques capable of identifying highly complex patterns in large datasets. Here, we provide a perspective and primer on deep learning applications for genome analysis. We discuss successful applications in the fields of regulatory genomics, variant calling and pathogenicity scores. We include general guidance for how to effectively use deep learning methods as well as a practical guide to tools and resources. This primer is accompanied by an interactive online tutorial.

Deep learning has made impressive recent advances in applications ranging from computer vision to natural-language processing. This primer discusses the main categories of deep learning methods and provides suggestions for how to effectively use deep learning in genomics. The primer is intended for bioinformaticians who are interested in applying deep learning approaches, and for genomicists and general biomedical researchers who seek a high-level understanding of this rapidly evolving field. Computer scientists may also use the primer as an introduction to the exciting applications of deep learning in genomics. However, we do not provide a survey of deep learning in the biomedical field, which has been broadly covered in recent reviews[1–5]. This paper is accompanied by an interactive tutorial that we have created for interested readers to build a convolutional neural network to discover DNA-binding motifs (see URLs).

Deep learning as a class of machine learning methods

Machine learning techniques have been extensively used in genomics research[3,6]. Machine learning tasks fall within two major categories: supervised and unsupervised. In supervised learning, the goal is predicting the label (classification) or response (regression) of each data point by using a provided set of labeled training examples. In unsupervised learning, such as clustering and principal component analysis, the goal is learning inherent patterns within the data themselves.

The ultimate goal in many machine learning tasks is to optimize model performance not on the available data (training performance) but instead on independent datasets (generalization performance). With this goal, data are randomly split into at least three subsets: training, validation and test sets. The training set is used for learning the model parameters (detailed discussion on parameter optimization in ref. [1]), the validation set is used to select the best model, and the test set is kept aside to estimate the generalization performance (Fig. 1). Machine learning must reach an appropriate balance between model flexibility and the amount of training data. An overly simple model will underfit and fail to let the data 'speak'. An overly flexible model will overfit to spurious patterns in the training data and will not generalize.

Large neural networks, a main form of deep learning, are a class of machine learning algorithms that can make predictions and perform dimensionality reduction. The key difference between deep learning and standard machine learning methods used in genomics—e.g., support vector machine and logistic regression—is that deep learning models have a higher capacity and are much more flexible. Typical deep learning models have millions of trainable parameters. However, this flexibility is a double-edged sword. With appropriately curated training data, deep learning can automatically learn features and patterns with less expert handcrafting. It also requires greater care to train on and to interpret the underlying biology. Box 1 summarizes the main messages of this primer on how to effectively use deep learning in genomics.

Setting up deep learning

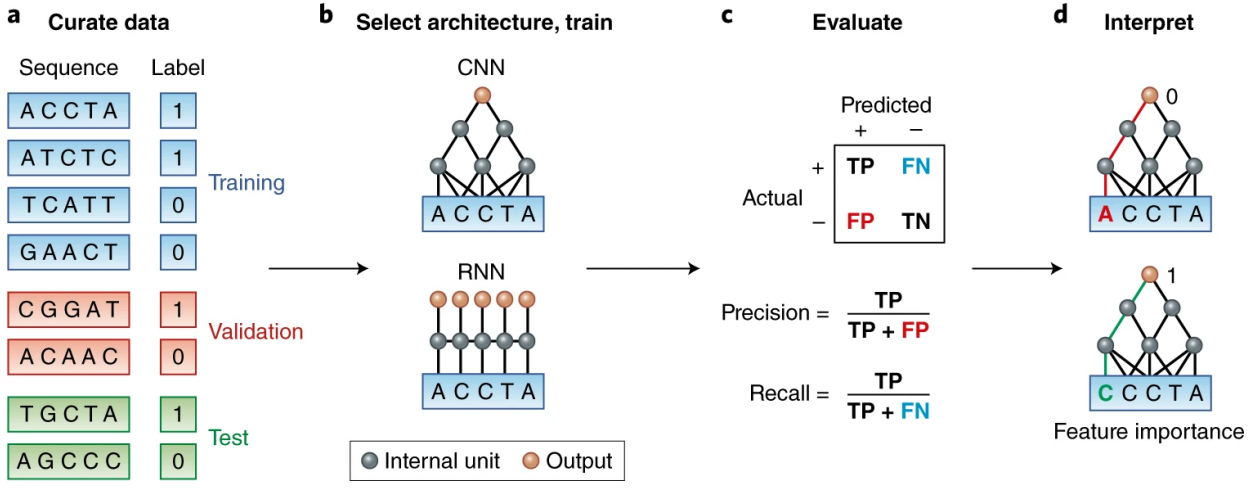
Deep learning is an umbrella term that refers to the recent advances in neural networks and the corresponding training platforms (e.g., TensorFlow and PyTorch). The starting point of a neural network is an artificial neuron, which takes as input a vector of real values and computes the weighted average of these values followed by a nonlinear transformation, which can be a simple threshold[7]. The weights are the parameters of the model that are learned during training. The power of neural networks stems from individual neurons being highly modular and composable, despite their simplicity[8]. The output of one neuron can be directly fed as input into other neurons. By composing neurons together, a neural network is created.

The input into a neural network is typically a matrix of real values. In genomics, the input might be a DNA sequence, in which the nucleotides A, C, T and G are encoded as [1,0,0,0], [0,1,0,0], [0,0,1,0] and [0,0,0,1]. Neurons that directly read in the data input are called the first, or input, layer. Layer two consists of neurons that read in the outputs of layer one, and so on for deeper layers, which are also referred to as hidden layers. The output of the neural network is the prediction of interest, e.g., whether the input DNA is an enhancer. Box 2 describes key terms and concepts in deep learning.

There are three common families of architectures for connecting neurons into a network: feed-forward, convolutional and recurrent. Feed-forward is the simplest architecture[7]. Every neuron of layer i is connected only to neurons of layer $i + 1$, and all the connection edges can have different weights. Feed-forward architecture is suitable for generic prediction problems when there are no special relations among the input data features.

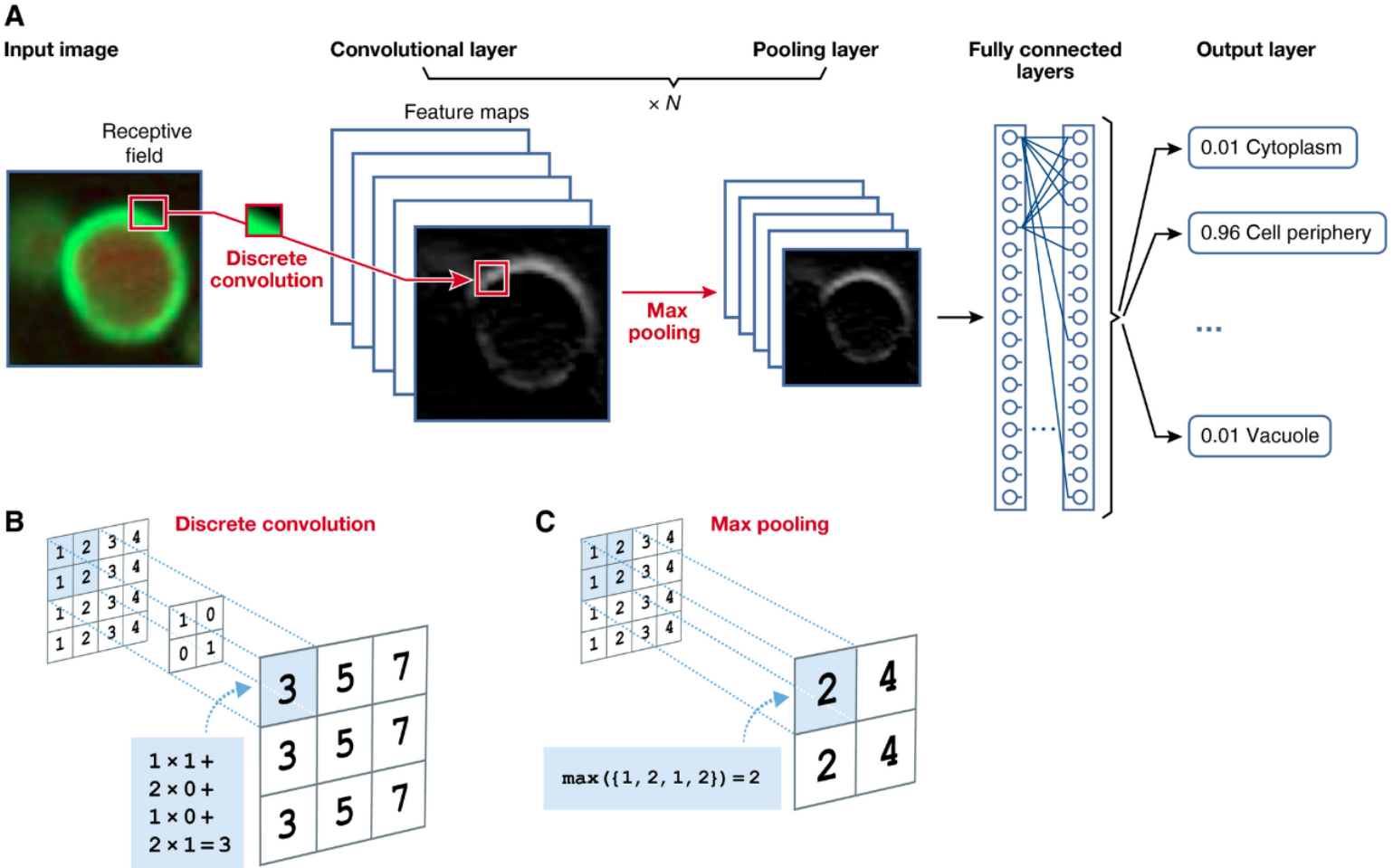
In a convolutional neural network (CNN), a neuron is scanned across the input matrix, and at each position of the input, the CNN

[1]Department of Biomedical Data Science, Stanford University, Palo Alto, CA, USA. [2]Chan-Zuckerberg Biohub, San Francisco, CA, USA. [3]Department of Electrical Engineering, Stanford University, Palo Alto, CA, USA. [4]Peltarion, Stockholm, Sweden. [5]Department of Learning, Informatics, Management and Ethics, Karolinska Institutet, Stockholm, Sweden. [6]Scripps Research Translational Institute, La Jolla, CA, USA. [7]Department of Integrative Structural and Computational Biology, The Scripps Research Institute, La Jolla, CA, USA. *e-mail: jamesz@stanford.edu; atelenti@scripps.edu



One-dimensional convolutional neural networks (CNNs) and Recurrent Neural Networks (example: LSTMs) are suitable for working with sequential data. Therefore, can be used for encoding DNA nucleotide sequences.

Convolutional Neural Networks (CNNs)



Angermueller et al., Molecular Systems Biology 2016

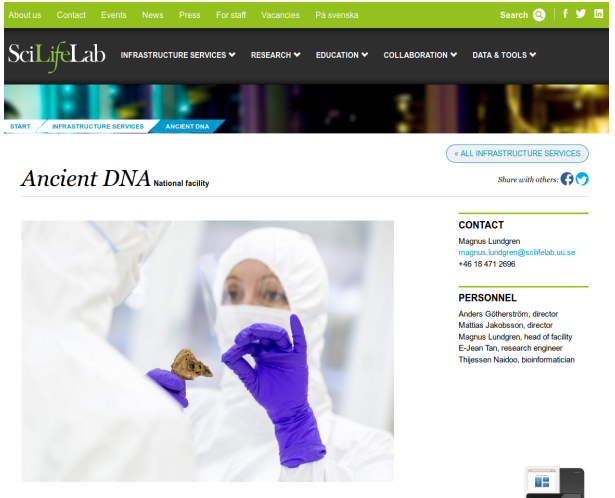
Introduction to ancient genomics

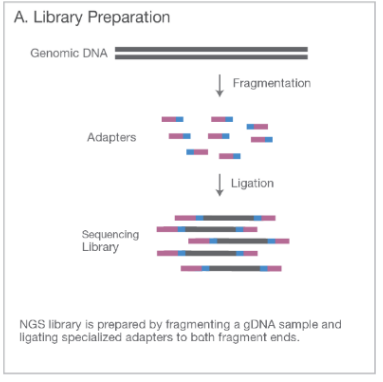
DNA from archaeological samples

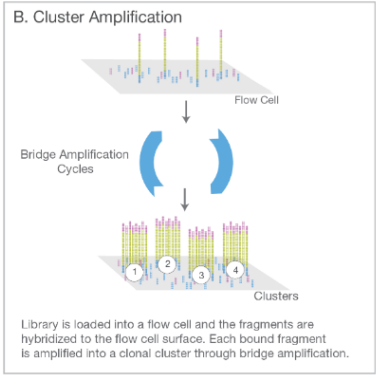
TTCAGGAAAACCCGATAAAAATTCTTTATTGGGGGAGGGGCTCAAACAAGAAAATAATCAACAAGTGGTGTCCAGAGTGGAGCCAGGG
CCTCCTGGGGACAGCAAGACTGCCTGGGGAGCGGGAAGCAGCTCCCCCGTCTCTGGGGGAGGCGTGGCTGGAGGGGAGGCTGGACCA
CAGGAGGGCAGCGCCCTGGGCAACCCTATGTAGATGAAGCTGCCGGAGAGGATCAAAGAACCAGACAGGAGGAAAGAGGCGGTGAAG
TCTCCTGTCTCATCCCTTAGGAAGCCTGAGGAGATGGGTAAGGGCATTTAGAAGCCTCGAACCCCAGGGGCAGAGGATAGTTGTAGGCA

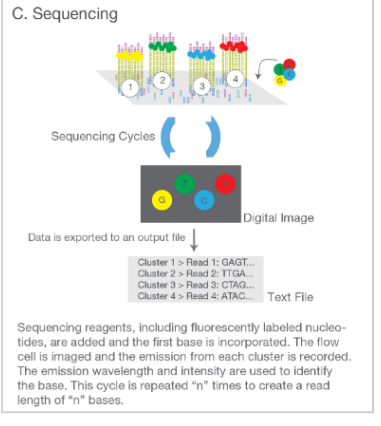
Peculiarity: very few samples with little amounts of DNA, hence hard for statistical analysis

Ancient DNA (aDNA) sequencing

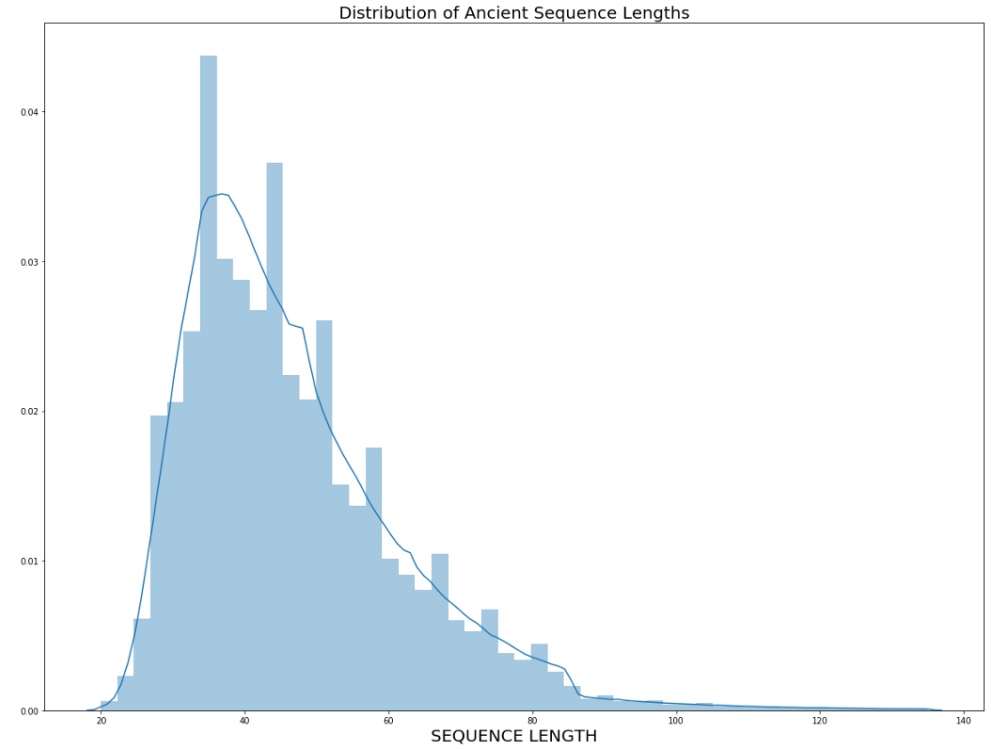




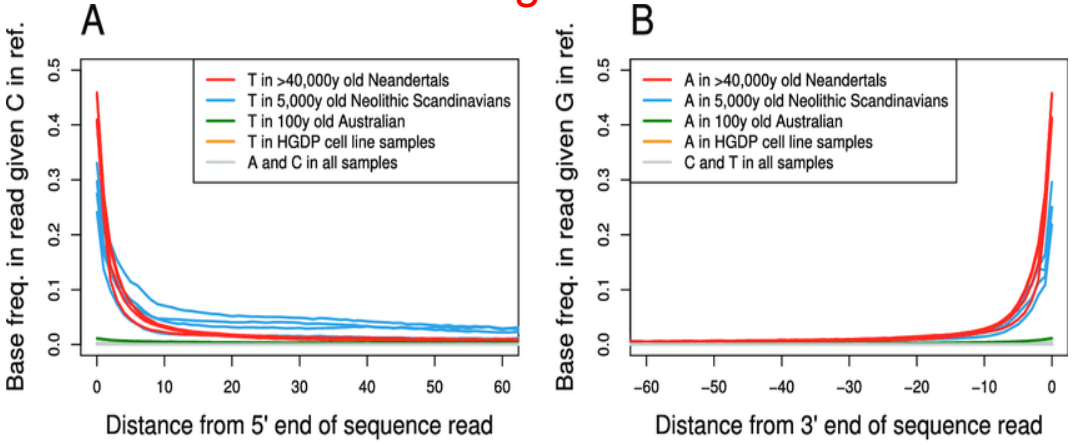




Ancient DNA (aDNA) degradation



Damage Pattern



Deamination process:

C → T

G → A

The problem of modern contamination

between enzymes encoded by structural genes originating from two different streptomycetes, rather than the results of the activation of latent genetic information in the recipient strains. (The latter mechanism is the most probable explanation for the three reports of the discovery of novel compounds through natural inter-strain matings (refs 18–20; discussed in ref. 21).) The evidence is most compelling for mederrhodin A, since, of the clones tested, pIJ2301, pIJ2315 and pIJ2316 (which led to mederrhodin A synthesis by AM-7161) all contain a complete transcription unit (absent from pIJ2304, pIJ2308, pIJ2312 and pIJ2317) which complements class V *act* mutants of *S. coelicolor* (F.M., unpublished); such mutants have recently been shown to be blocked in the corresponding hydroxylation involved in actinorhodin biosynthesis (S. P. Cole and H.G.F., unpublished). A true metabolic cooperation between the actinorhodin and granaticin biosynthetic enzymes is indicated also by complementation of the B1140 mutant by pIJ2308, but not by the other clones tested. B1140 can act as a secretor in co-synthesis tests with *S. coelicolor act* mutants of classes I, III and VII, but not of classes IV, V and VI[22], suggesting that its block is equivalent to that of class IV *act* mutants. Significantly, pIJ2308

Molecular cloning of Ancient Egyptian mummy DNA

Svante Pääbo

Department of Cell Research, The Wallenberg Laboratory, University of Uppsala, Box 562, S-75122 Uppsala, Sweden and Institute of Egyptology, Gustavianum, University of Uppsala, S-75120 Uppsala, Sweden

Artificial mummification was practised in Egypt from ~2600 BC until the fourth century AD. Because of the dry Egyptian climate, however, there are also many natural mummies preserved from earlier as well as later times. To elucidate whether this unique source of ancient human remains can be used for molecular genetic analyses, 23 mummies were investigated for DNA content. One 2,400-yr-old mummy of a child was found to contain DNA that could be molecularly cloned in a plasmid vector. I report here that one such clone contains two members of the *Alu* family of human

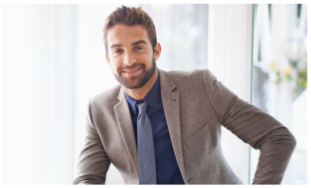
Fig. 2 DNA sequence of part of the mummy clone pMUM2:9. The 9-bp direct repeats flanking the *Alu* sequence are underlined. In the *Alu* consensus sequence[8] only nucleotides differing from the pMUM2:9 sequence are specified.

Methods. DNA was prepared from 1.6 g mummy tissue, essentially according to the protocol described previously[5]. The DNA was made blunt-ended with *Escherichia coli* DNA polymerase (Klenow fragment) (Pharmacia P-L Biochemicals) according to the manufacturer's instructions and in the presence of trace amounts of radioactively labelled nucleotides. The sample was then size-fractionated by gel filtration on a G-50 column (Pharmacia, Uppsala). Excluded fractions were pooled and ethanol-precipitated and 25 ng of this material cloned in a *Sma*I-digested and alkaline phosphatase-treated pUC8 plasmid[6]. Then, 700 of the white clones were transferred to nitrocellulose filters and screened[20] with a nick-translated[7] 550-bp *Bgl*II/*Sph*I fragment from a HLA-DR pseudogene[21], which contains an *Alu* repeat. The strongly hybridizing clone pMUM2:9 was isolated and restriction-mapped. Two *Alu* repeats were identified by Southern hybridization[9]. One of the *Alu* repeats as well as 500 bp of flanking DNA were sequenced according to the Maxam and Gilbert procedure[22] after labelling of the *Sph*I and *Nde*I restriction sites indicated.

Deep Learning for Ancient Genomics

The problem of modern contamination

Modern Sequences



AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

Ancient Sequences

AGCCATATGCATGCATCGTAGTC

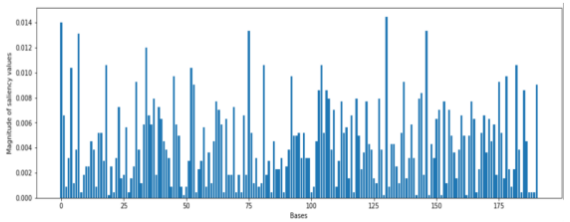
AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

AGCCATATGCATGCATCGTAGTC

Compare and learn DNA patterns (k-mers) that separate ancient and modern sequences



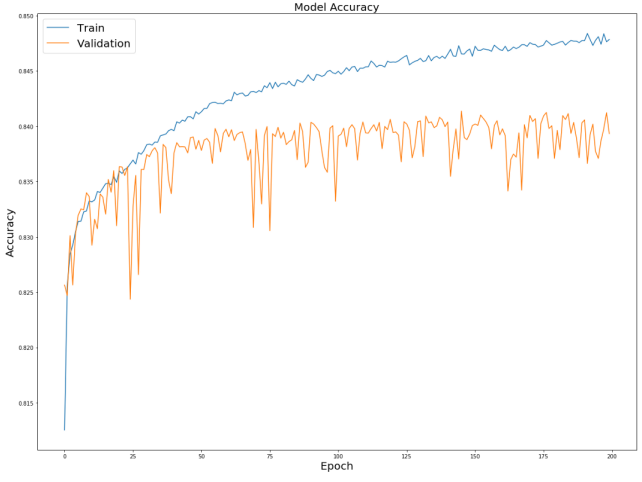
Encoding DNA sequence and training CNN

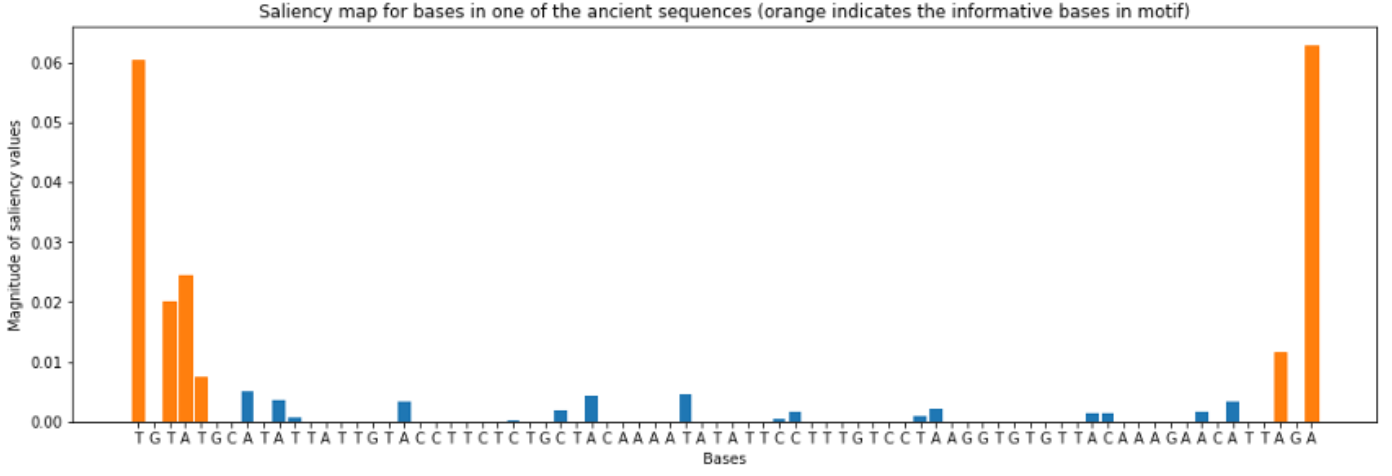
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# The LabelEncoder encodes a sequence of bases as a sequence of integers: 0, 1, 2 and 3
integer_encoder = LabelEncoder()
# The OneHotEncoder converts an array of integers to a sparse matrix where
# each row corresponds to one possible value of each feature, i.e. only 01 and 1 are present in the matrix
one_hot_encoder = OneHotEncoder()
input_features = []

for sequence in sequences:
  integer_encoded = integer_encoder.fit_transform(list(sequence))
  integer_encoded = np.array(integer_encoded).reshape(-1, 1)
  one_hot_encoded = one_hot_encoder.fit_transform(integer_encoded)
  input_features.append(one_hot_encoded.toarray())

np.set_printoptions(threshold=40)
#print(input_features.shape)
input_features = np.stack(input_features)
print("Example sequence\n-----------------------")
print('DNA Sequence #1:\n',sequences[0][:10],'...',sequences[0][-10:])
print('One hot encoding of Sequence #1:\n',input_features[0].T)
```

```
Example sequence
-----------------------
DNA Sequence #1:
 AAATCATCAG ... CTCCAAAACT
One hot encoding of Sequence #1:
 [[ 1.  1.  1. ...,  1.  0.  0.]
 [ 0.  0.  0. ...,  0.  1.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  1.]]
```

```
from keras.optimizers import SGD, Adam, Adadelta
from keras.layers import Conv1D, Dense, MaxPooling1D, Flatten, Dropout, BatchNormalization, Activation
from keras.models import Sequential
from keras.regularizers import l1

model = Sequential()

model.add(Conv1D(filters=32, kernel_size=5, padding='same', kernel_initializer= 'he_uniform',
                 input_shape=(train_features.shape[1], 4)))
model.add(Activation("relu"))
model.add(Conv1D(filters=32, kernel_size=5, padding='same', kernel_initializer= 'he_uniform'))
model.add(Activation("relu"))
model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(16, kernel_initializer= 'he_uniform'))
model.add(Activation("relu"))
model.add(Dense(2, activation='softmax'))

epochs = 20
lrate = 0.01
decay = lrate / epochs
sgd = SGD(lr = lrate, momentum = 0.9, decay = decay, nesterov = False)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['binary_accuracy'])
#model.compile(loss='binary_crossentropy', optimizer=Adam(lr = lrate), metrics=['binary_accuracy'])
model.summary()
```

```
Using TensorFlow backend.
```





DNA Sequence as a Text:
Natural Language Processing Approach

What we usually mean by texts

Editing Wikipedia articles on

Medicine

Editing Wikipedia can be daunting for newbies, especially if you're contributing to Wikipedia for the first time as a class assignment. This guide is designed to assist students who have been assigned to contribute biomedical-related content to Wikipedia. Here's what other editors will expect you to know.

Be accurate

You're editing a resource millions of people use to make medical decisions, so it's vitally important to be accurate. Wikipedia is used more for medical information than the websites for WebMD, NIH, and the WHO. But with great power comes great responsibility!

Understand the guidelines

Wikipedia editors in the medicine area have developed additional guidelines to ensure that the content on Wikipedia is medically sound. Take extra time to read and understand these guidelines. When you edit an article, ensure your changes meet these special requirements. If not, your work is likely to be undone by other editors as they clean up after you. (Can takes valuable volunteer time away from creating content. If you're not comfortable working under these guidelines, talk to your instructor about an alternative off-wiki assignment.

Engage with editors

Part of the Wikipedia experience is receiving and responding to feedback from other editors. Do not submit your content on the last day, then leave Wikipedia! Real human volunteers from the Wikipedia community will likely read and respond to it, and it would be polite for you to acknowledge the time they volunteer to polish your work! Everything submitted to Wikipedia is reviewed by multiple, real humans! You may not get a comment, but if you do, please acknowledge it.

Watch out for close paraphrasing

Plagiarizing or close paraphrasing is never okay on Wikipedia and is a violation of your university's academic honor code. It's even worse on Wikipedia, as valuable volunteer time that could be used to create good content is instead used to clean up plagiarized work.

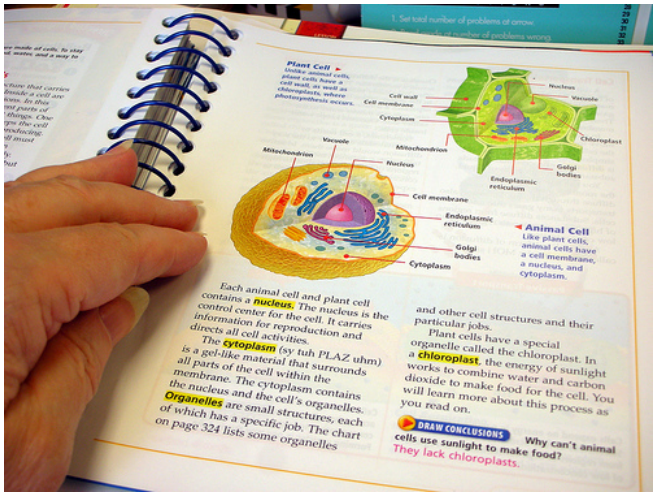
If you plagiarize or too closely paraphrase on Wikipedia, it is extremely likely that you'll be caught by other editors and there will be an online record of your plagiarism tied to your permanent online record.

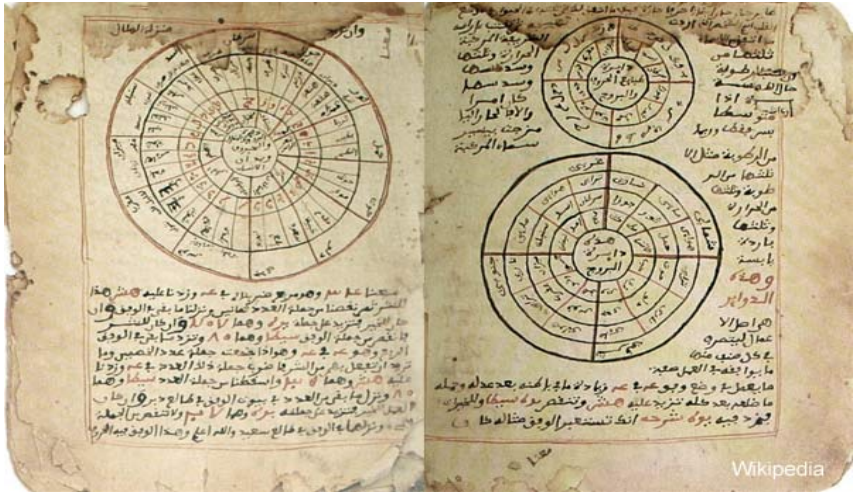
Note that even educational materials from organizations like the WHO and abstracts of articles in PubMed are under copyright and cannot be copied. Write them in your own words whenever possible. If you aren't clear on what close paraphrasing is, visit your university's writing center.

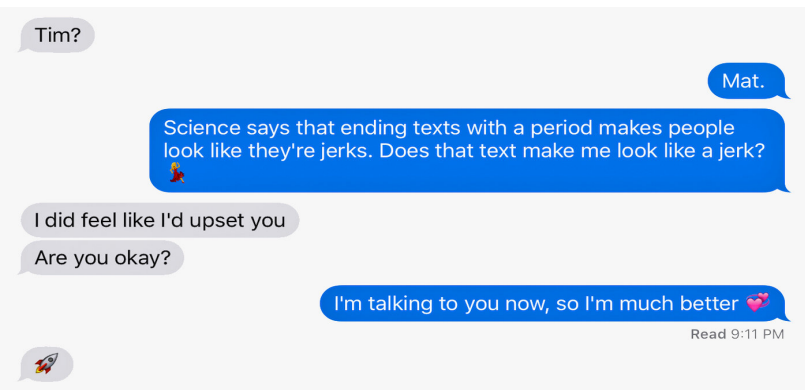
Scared? Don't be!

Everybody on Wikipedia wants to make the best encyclopedia they can. Take the time to understand the rules, and soon you'll be contributing to a valuable resource you use on a daily basis!

Wiki Edu







What bioinformaticians mean by texts

This is a text, so we can use Natural Language Processing (NLP)

Where are sentences? Where are words?

Convert Sequence to Sentence via Kmers

Sequence

GATCGTAC

Kmers

GATCGTAC

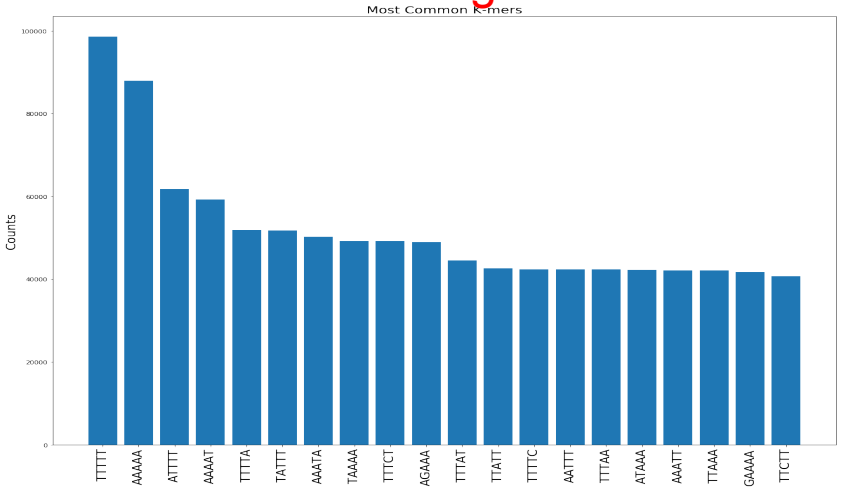
GATCGTAC

GATCGTAC

GATCGTAC

Functional elements vs. junk DNA

NLP: Bag of Words



Sentence / Text

GATCG ATCGT TCGTA CGTAC

word word word word

Visualize Vocabulary: Word2Vec

```
import warnings
warnings.filterwarnings('ignore')

from gensim.models import Word2Vec
model = Word2Vec(sentences, min_count = 2, workers = 4)
print(model)
```

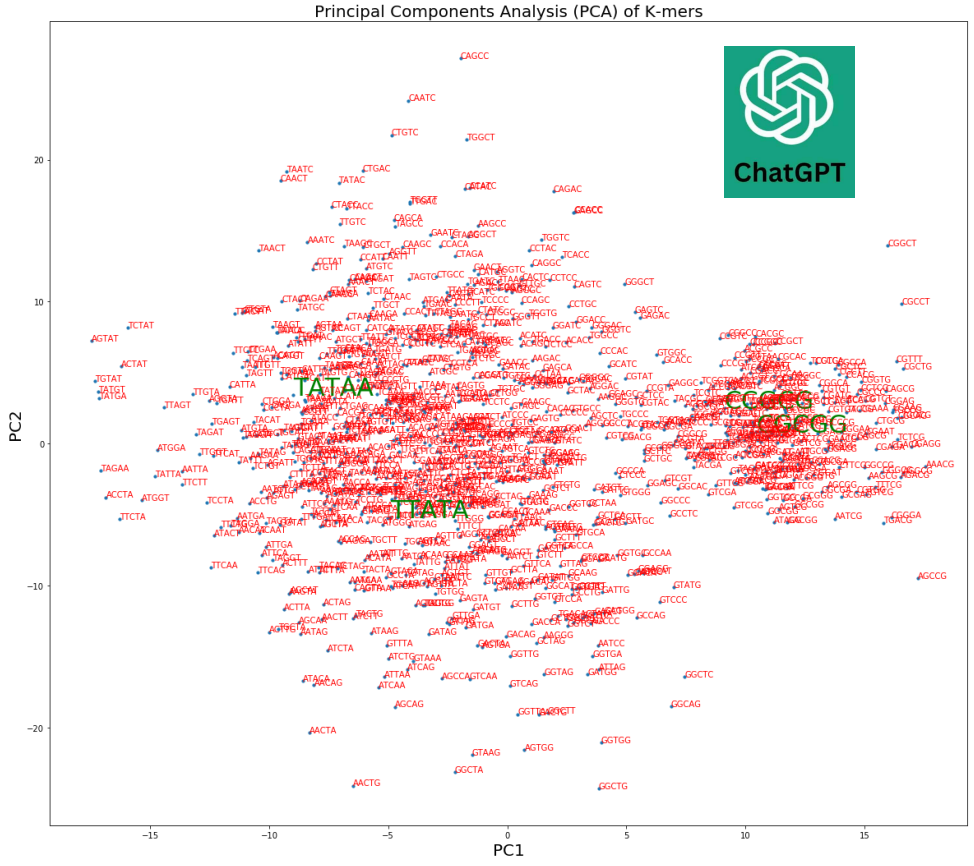
```
Word2Vec(vocab=1024, size=100, alpha=0.025)
```

```
X = model[model.wv.vocab]
X.shape
```

```
(1024, 100)
```

Now each word is one observation, this observation has 100 coordinates, i.e. the default number of latent variables for word2vec. Next we can try to use the constructed word vectors and visualize the k-mers space using PCA and UMAP.

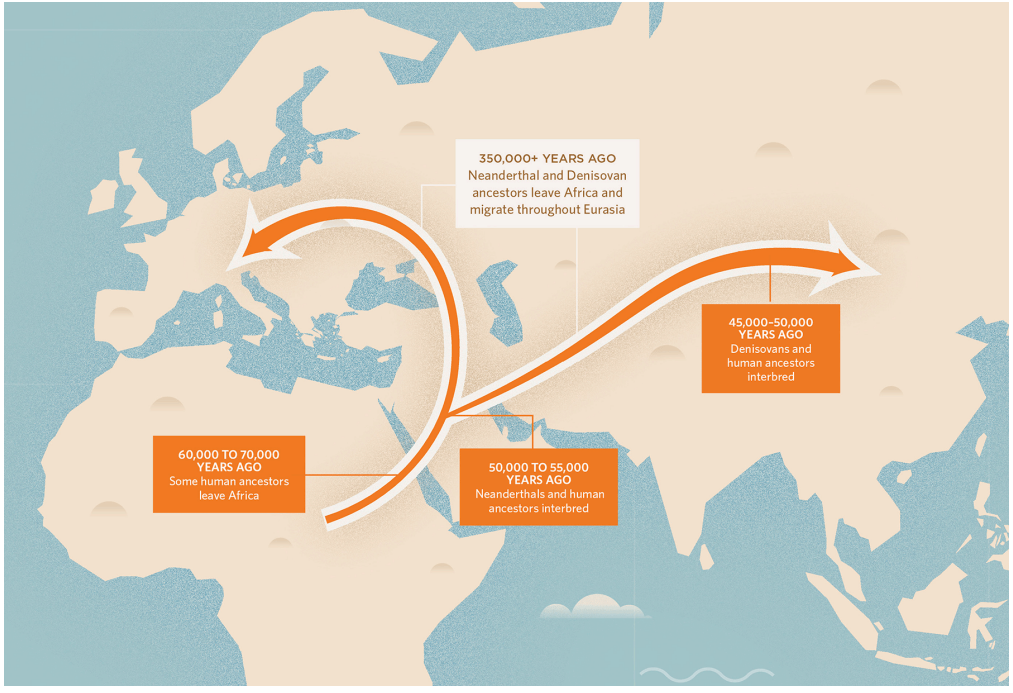
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
X = model[model.wv.vocab]
pca = PCA(n_components = 2)
result = pca.fit_transform(X)

plt.figure(figsize=(20,18))
plt.scatter(result[:, 0], result[:, 1], s = 10, cmap = 'tab10')
plt.title('Principal Components Analysis (PCA) of K-mers', fontsize = 20)
plt.xlabel("PC1", fontsize = 20)
plt.ylabel("PC2", fontsize = 20)
words = list(model.wv.vocab)
for i, word in enumerate(words):
    if word == 'CCGCG':
        plt.text(result[i, 0], result[i, 1], word, fontsize = 30, c = 'green')
    elif word == 'CGCGG':
        plt.text(result[i, 0], result[i, 1], word, fontsize = 30, c = 'green')
    elif word == 'TTATA':
        plt.text(result[i, 0], result[i, 1], word, fontsize = 30, c = 'green')
    elif word == 'TATAA':
        plt.text(result[i, 0], result[i, 1], word, fontsize = 30, c = 'green')
    else:
        plt.text(result[i, 0], result[i, 1], word, fontsize = 10, c = 'red')
plt.show()
```

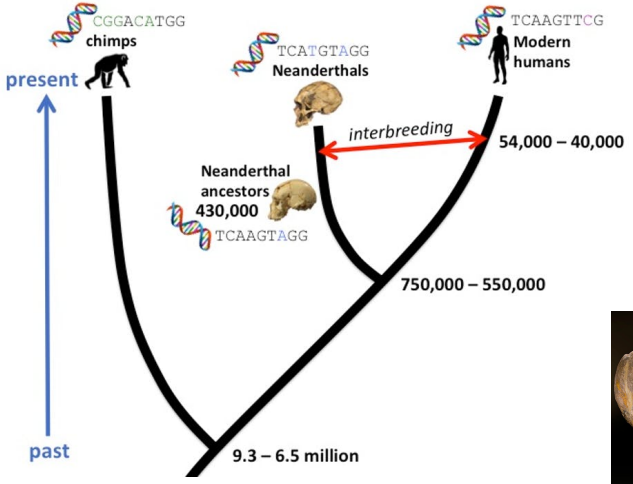


GC-rich and AT-rich 5-mers seem to form separate clusters of nucleotide “words”

Applications for detecting Neanderthal DNA in modern genomes

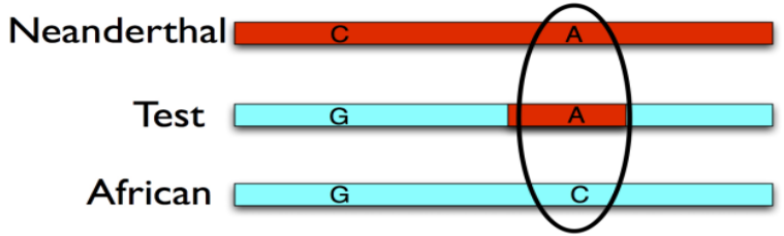
Neanderthal Genomics

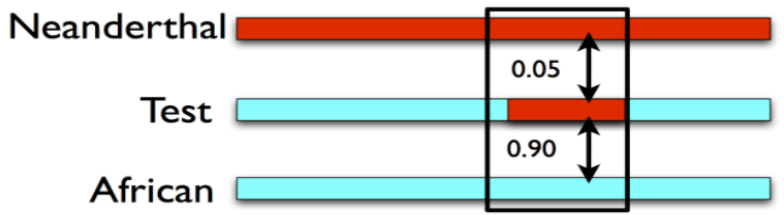


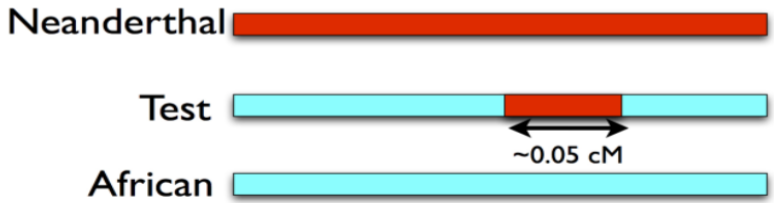


2010 Draft Neanderthal Genome Sequenced

Segments of Neanderthal introgression







Conditional Random Fields (CRF): David Reich et al., Nature 2014, 2016
S* and S' statistics: Benjamin Vernot and Joshua Akey, Science 2014, 2016

Sentiments Analysis: Bag of Words

Shallow MLP: 82% accuracy

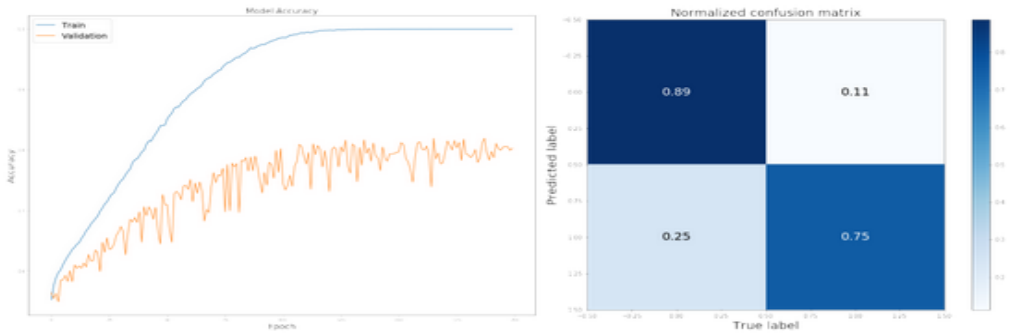
```
from keras.models import Sequential
from keras.regularizers import l2, l1
from keras.callbacks import ModelCheckpoint
from keras.optimizers import SGD, Adam, Adadelta
from keras.layers import Dense, Flatten, Dropout

model = Sequential()
model.add(Dense(3000, input_shape = (X.shape[1], ), activation = 'sigmoid',
                kernel_regularizer = l1(0.00001)))
model.add(Dense(1, activation = 'sigmoid'))
sgd = SGD(lr = 0.0001, momentum = 0.9, nesterov = False)
model.compile(loss = 'binary_crossentropy', optimizer = sgd, metrics = ['binary_accuracy'])
checkpoint = ModelCheckpoint("weights.best.hdf5", monitor='val_binary_accuracy', verbose=1,
                             save_best_only = True, mode = 'max')
history = model.fit(X_train, y_train,
                    epochs = 200, verbose = 1, validation_split = 0.2, batch_size = 32,
                    shuffle = True, callbacks = [checkpoint])
```

mlp.py hosted with ❤ by GitHub view raw

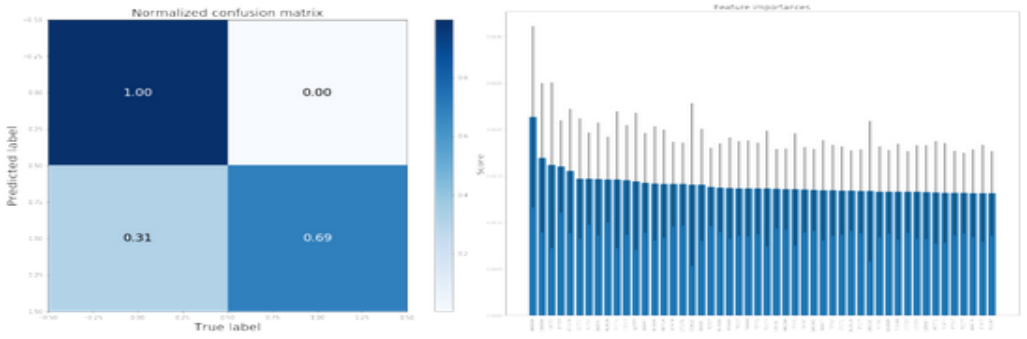
Random Forest: 85% accuracy

```
1  import pickle
2  import matplotlib.pyplot as plt
3  from sklearn.ensemble import RandomForestClassifier
4
5  classifier = RandomForestClassifier(n_estimators = 500)
6  classifier.fit(X_train, y_train)
7  y_pred = classifier.predict(X_test)
8  pickle.dump(classifier, open('RF_model_Neand_Intr_vs_Depl.sav', 'wb'))
9
10 importances = classifier.feature_importances_
11 std = np.std([tree.feature_importances_ for tree in classifier.estimators_], axis = 0)
12 indices = np.argsort(importances)[::-1]
13 plt.title("Feature importances", fontsize = 20)
14 plt.bar(range(X_train.shape[1])[0:50], importances[indices][0:50],
15         yerr = std[indices][0:50], align = "center")
16 plt.xticks(rotation = 90); plt.ylabel('Score', fontsize = 20)
17 plt.xticks(range(X_train.shape[1])[0:50], np.array(names)[indices][0:50])
```

RF.py hosted with ❤ by GitHub view raw



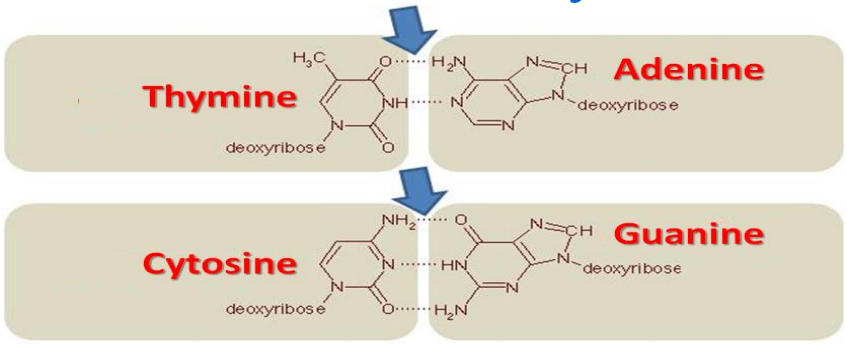
MLP Accuracy training curve (left) and confusion matrix of evaluation on test data set (right)

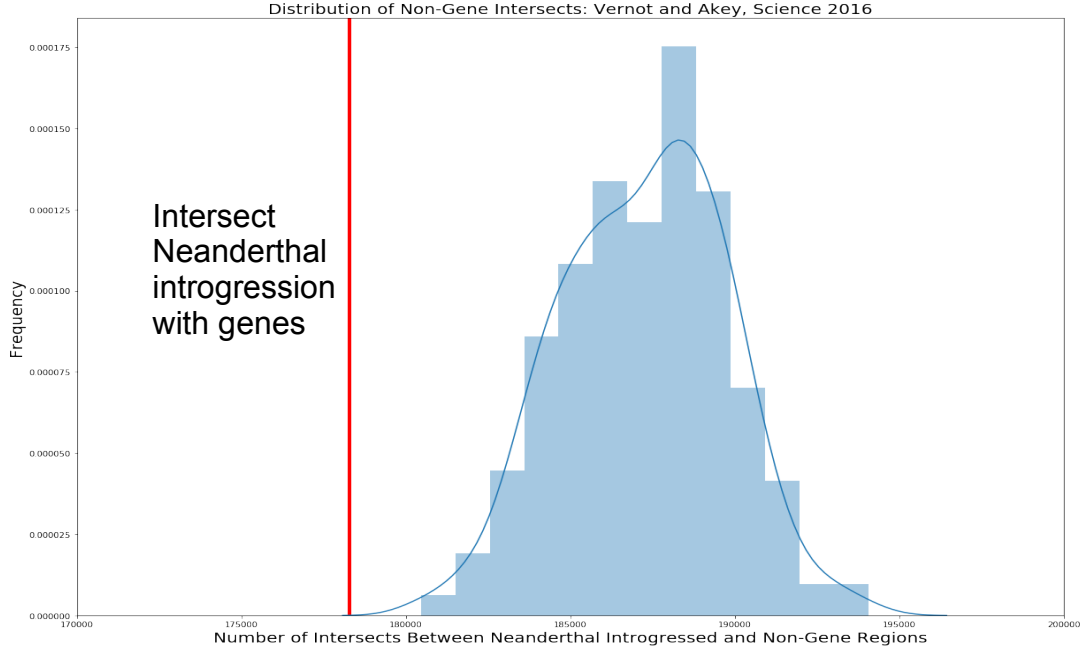


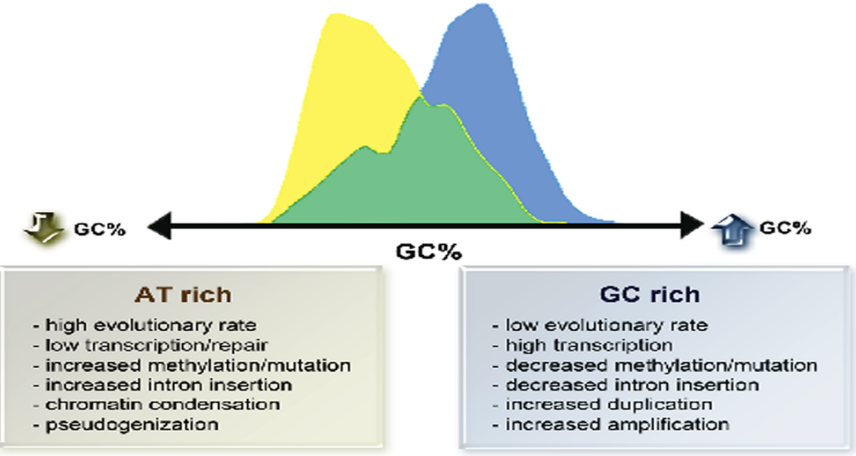
Random Forest classifier confusion matrix (left) and feature importances dominated by AT-rich k-mers (right)

AAAAA, CAAAA, CATTT and TTTTT are most predictive, i.e. **AT-rich (gene depletion)**

Evolution seems to eliminate Neanderthal ancestry from modern human genes

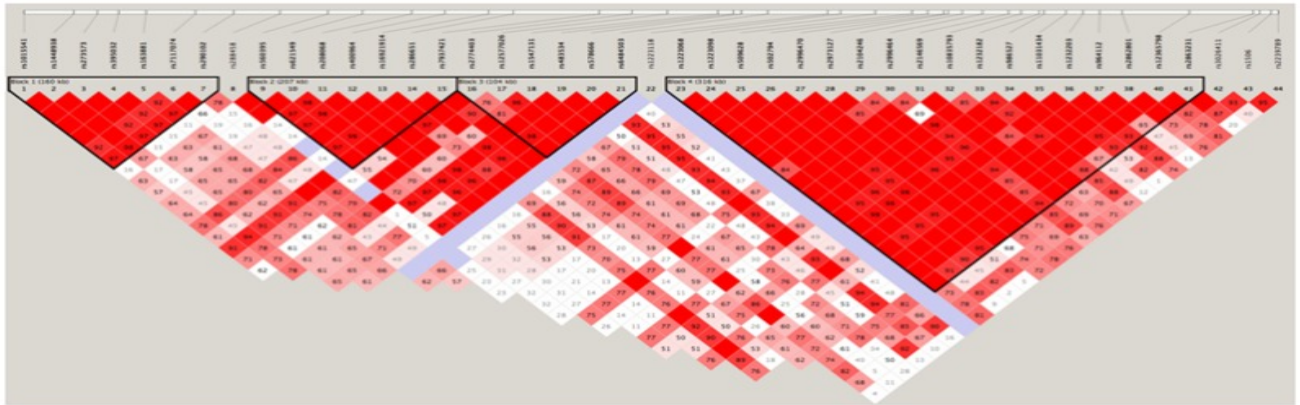


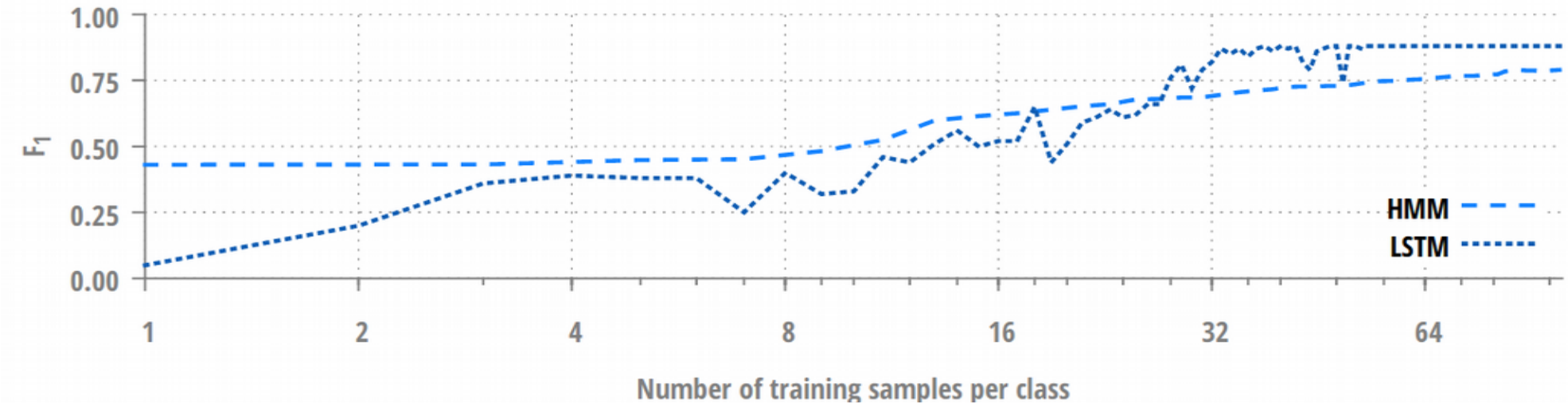




It seems that breeding with Neanderthals was not beneficial for modern humans?

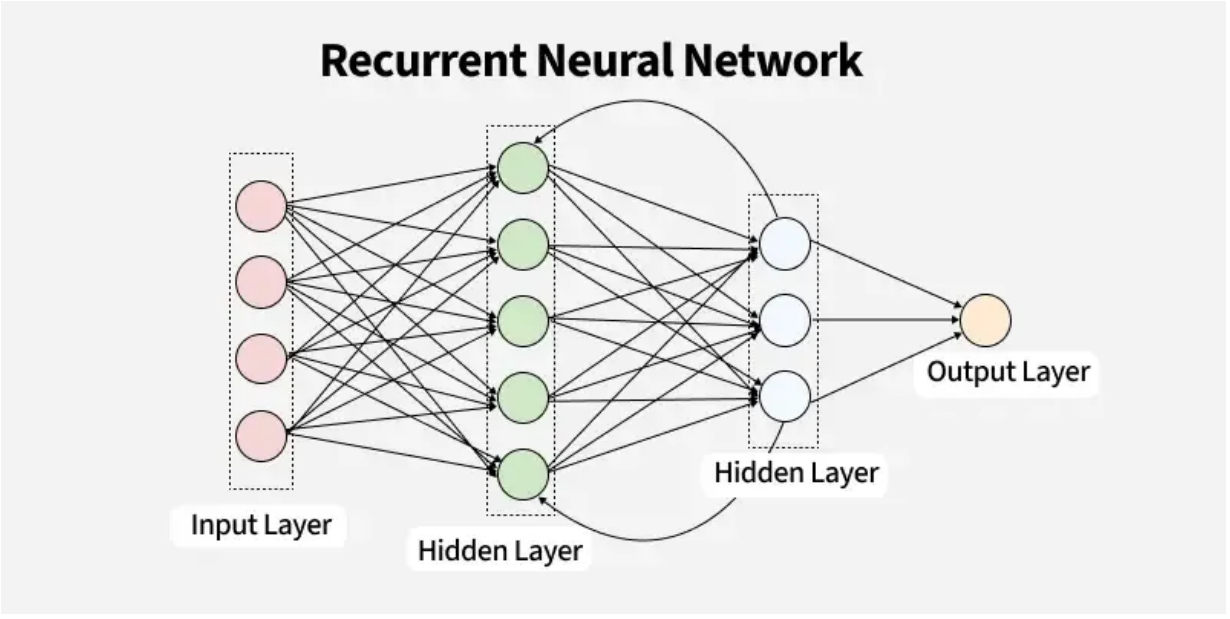
Long-Memory vs Memoryless Algorithms

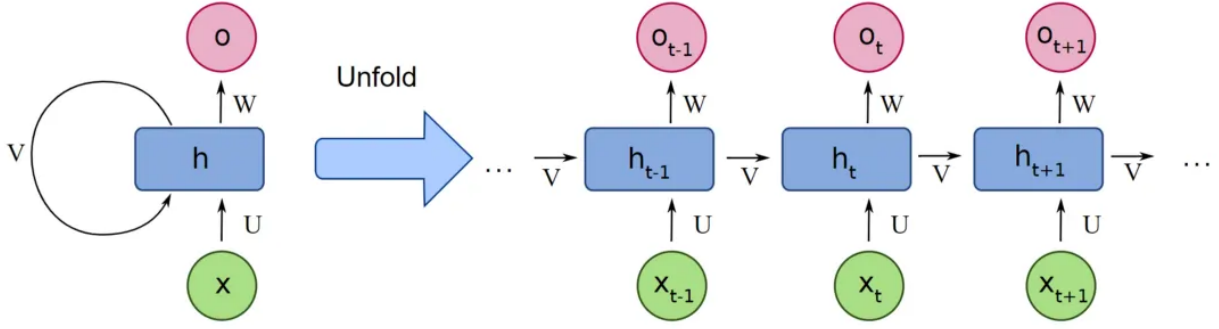




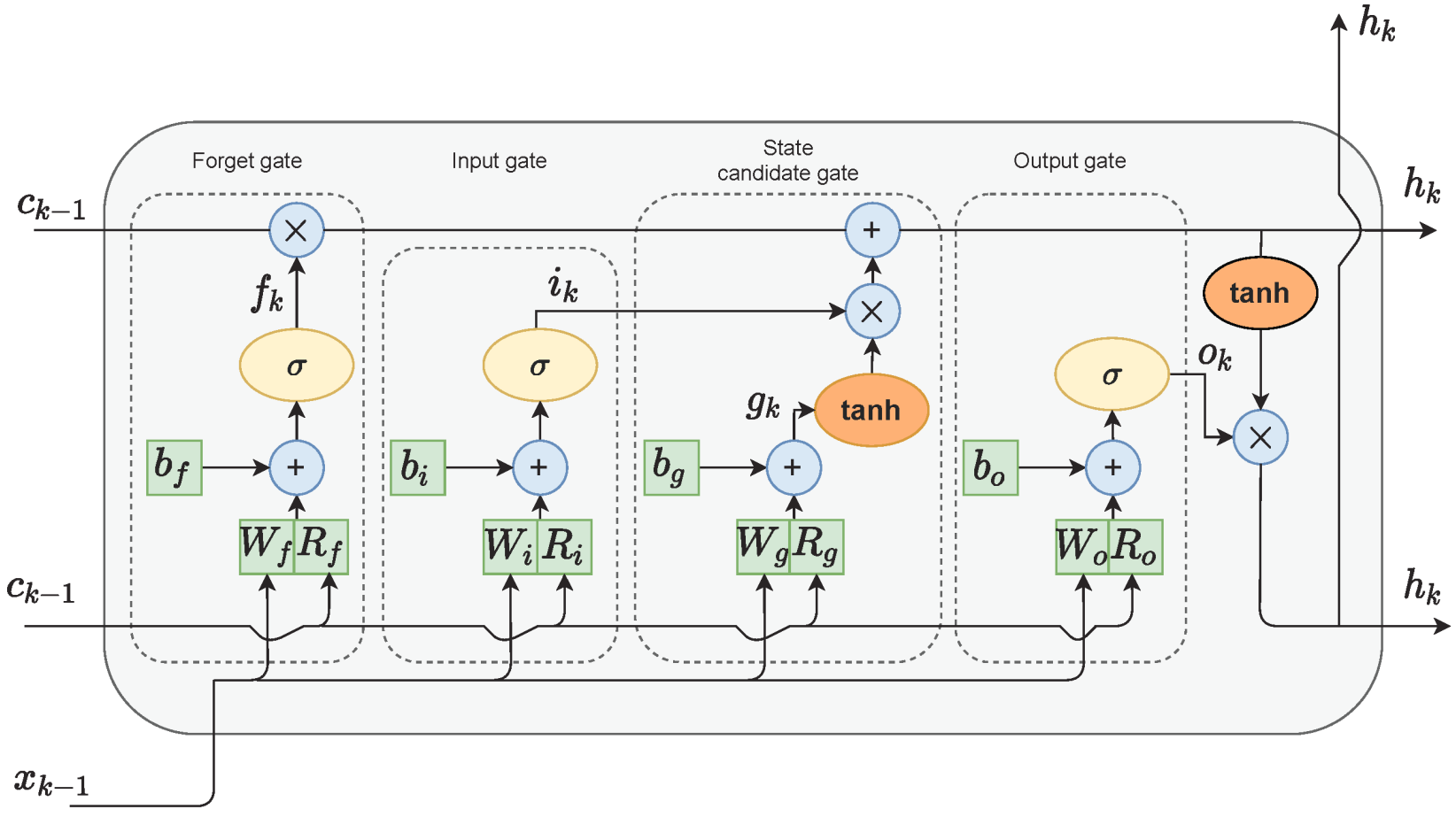
1) Long memory algorithms are better suited for capturing the Linkage Disequilibrium
2) Tones of training data: human genome can be sliced into mi(bi)llions of segments

Recurrent Neural Network (RNN)





Long Short-Term Memory (LSTM)



Bi-Directional LSTM with Embedding

We end up with **737 340 sentences** that belong to two classes: Neanderthal introgressed and depleted. The next step is to one-hot-encode the sentences via converting the k-mers / words into **integers** using the **Tokenizer** class in Python. Note that padding is not really needed in our case as all sentences are of the same length, **191 k-mers long**, but I include it here for generality.

```python
import numpy as np
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences

merge_texts = intr_texts + depl_texts
labels = list(np.ones(len(intr_texts))) + list(np.zeros(len(depl_texts)))

tokenizer = Tokenizer()
tokenizer.fit_on_texts(merge_texts)
encoded_docs = tokenizer.texts_to_sequences(merge_texts)
max_length = max([len(s.split()) for s in merge_texts])
X = pad_sequences(encoded_docs, maxlen = max_length, padding = 'post')

X_train,X_test,y_train,y_test = train_test_split(X,labels,
                                                 test_size=0.20,random_state=42)
vocab_size = len(tokenizer.word_index) + 1
```

prepare_data.py hosted with ❤ by GitHub — view raw

The vocabulary size was 964 114 in our case which is less than 4^10 = 1 048 576 implying that not all possible 10-mers built out of 4 characters are present within the sequences. Finally, we define a **Sequential model** starting with the **Keras Embedding Layer** that learns **Word Embeddings** while classifying the sequences, followed by a **Bidirectional LSTM** and a Dense layers.

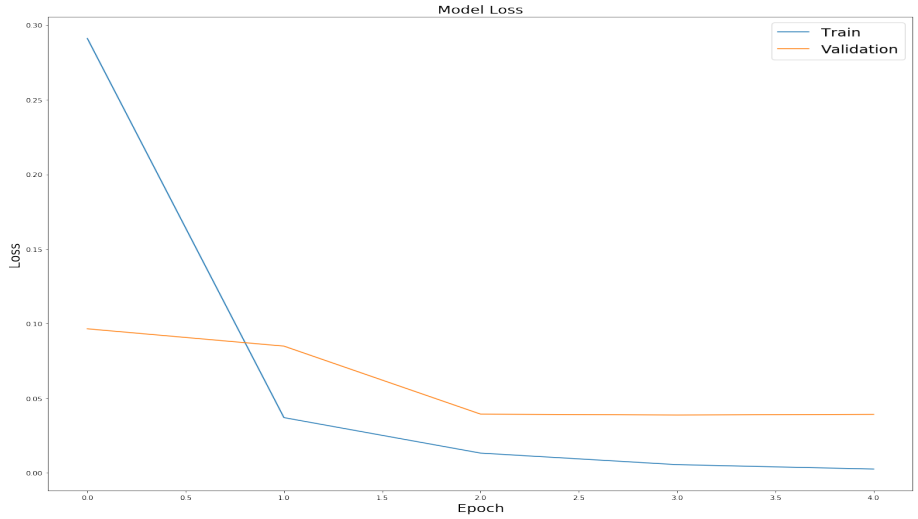
```python
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint
from keras.optimizers import SGD, Adam, Adadelta, RMSprop
from keras.layers import Embedding, LSTM, Dense, Bidirectional

model = Sequential()
model.add(Embedding(vocab_size, 10))
model.add(Bidirectional(LSTM(10)))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

epochs = 5
model.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
checkpoint=ModelCheckpoint("LSTM.weights.best.hdf5", monitor='val_acc',verbose = 1,
                           save_best_only = True, mode = 'max')
model.summary()
```

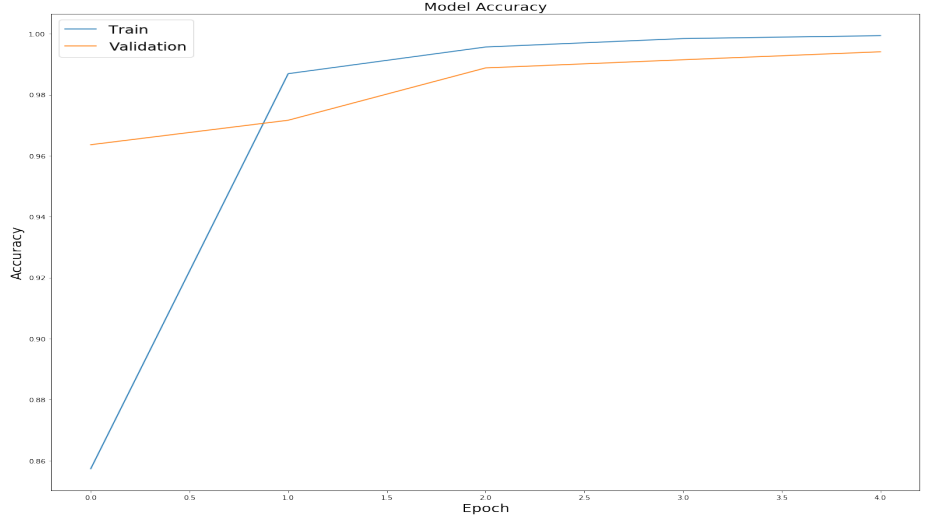
lstm.py hosted with ❤ by GitHub — view raw

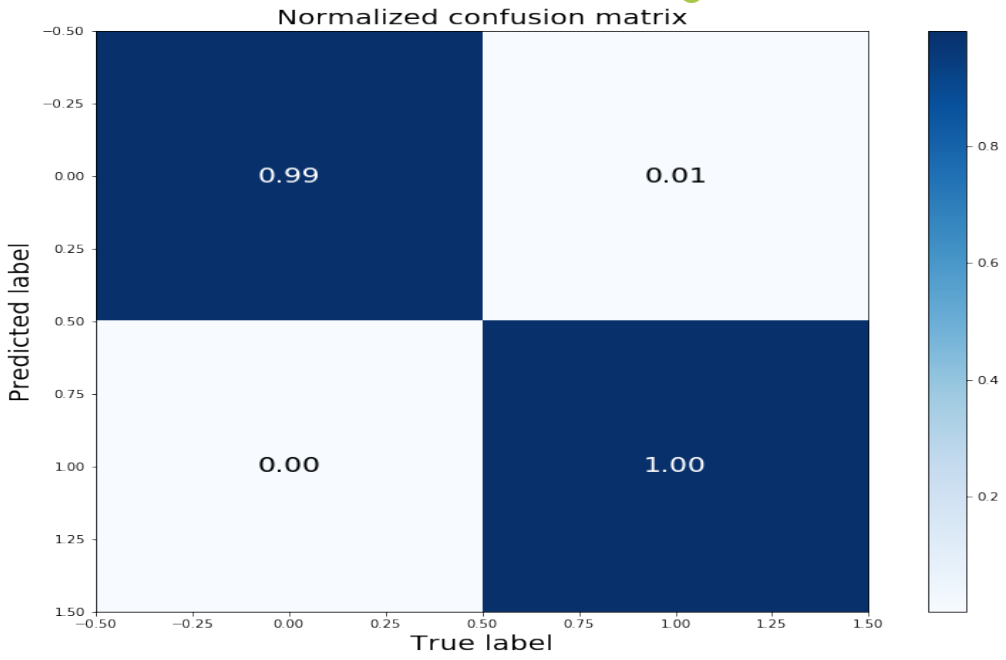
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 10)          9641140
_________________________________________________________________
bidirectional_2 (Bidirection (None, 20)                1680
_________________________________________________________________
dense_3 (Dense)              (None, 10)                210
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 9,643,041
Trainable params: 9,643,041
Non-trainable params: 0
_________________________________________________________________
None

Train on 471897 samples, validate on 117975 samples
Epoch 1/5
471897/471897 [==============================] - 3035s 6ms/step - loss: 0.2911 - acc: 0.8573 - val_loss: 0.0965 - v
al_acc: 0.9636

Epoch 00001: val_acc improved from -inf to 0.96362, saving model to weights.best.hdf5
Epoch 2/5
471897/471897 [==============================] - 5225s 11ms/step - loss: 0.0371 - acc: 0.9869 - val_loss: 0.0850 -
val_acc: 0.9716

Epoch 00002: val_acc improved from 0.96362 to 0.97165, saving model to weights.best.hdf5
Epoch 3/5
471897/471897 [==============================] - 3116s 7ms/step - loss: 0.0133 - acc: 0.9957 - val_loss: 0.0394 - v
al_acc: 0.9888

Epoch 00003: val_acc improved from 0.97165 to 0.98881, saving model to weights.best.hdf5
Epoch 4/5
471897/471897 [==============================] - 3172s 7ms/step - loss: 0.0055 - acc: 0.9984 - val_loss: 0.0388 - v
al_acc: 0.9915

Epoch 00004: val_acc improved from 0.98881 to 0.99149, saving model to weights.best.hdf5
Epoch 5/5
471897/471897 [==============================] - 3160s 7ms/step - loss: 0.0026 - acc: 0.9994 - val_loss: 0.0393 - v
al_acc: 0.9941

Epoch 00005: val_acc improved from 0.99149 to 0.99411, saving model to weights.best.hdf5
```

N_length x N_sequences = RAM of my laptop

Train LSTM: Visualizing Embeddings







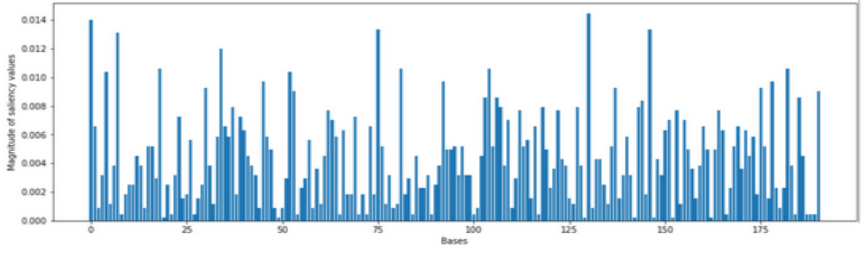
Visualizing Word Embeddings

Interpret LSTM: Feature Importances

One way to build feature importances for a neural network is to impose some **perturbation** on the input data and monitor the **variation** in the accuracy of prediction at the output of the network. Here, we would like to find the **most informative k-mers**, however, in our case the input matrix X had dimensions (737340, 191), where the first dimension represented the number of training examples for the neural network, and the second dimension corresponded to the number of words in each sentence / sequence. This implies that an **index** (or **position** in sentence) of each word / k-mer was the **feature** for the input matrix X. Therefore if we shuffle each of the 191 features, one at a time across the 737340 samples, and check the decrease in accuracy compared to the un-perturbed input matrix, we can rank the features by their importances for the final prediction. In our case, ranking features is equivalent to determining the most **important positions** of words / k-mers across all sentences / sequences.

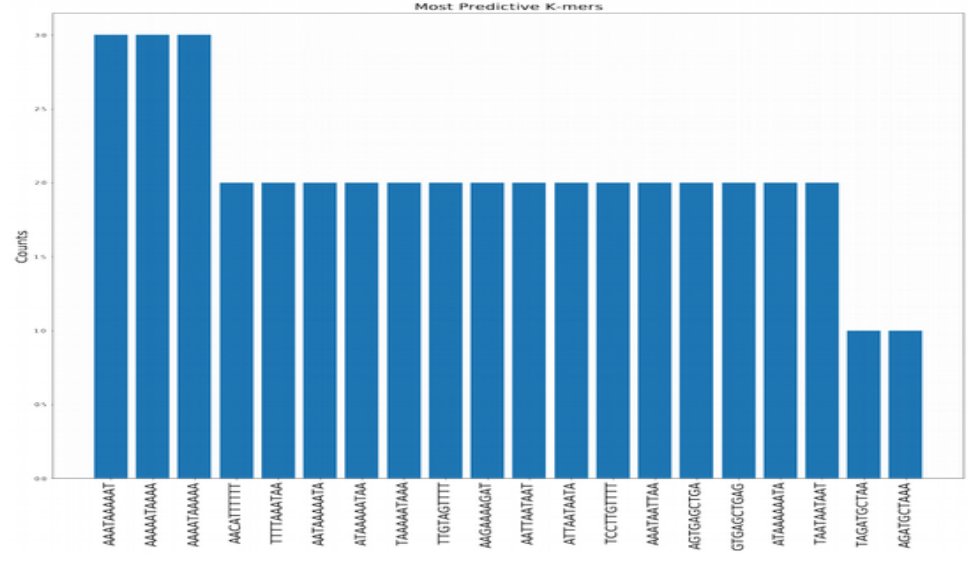
```python
import matplotlib.pyplot as plt

perm_scores = []
for i in range(X_test.shape[1]):
    X_test_perm = X_test
    X_test_perm[:,i] = random.sample(list(X_test[:,i]), len(list(X_test[:,i])))
    perm_scores.append(abs(model.evaluate(X_test_perm, y_test, verbose = 0)[1]*100 -
                           model.evaluate(X_test, y_test, verbose = 0)[1]*100))

plt.figure(figsize=[16,5])
barlist = plt.bar(np.arange(len(perm_scores)), perm_scores)
plt.xlabel('Bases'); plt.ylabel('Magnitude of saliency values')
```

perm_X.py hosted with ❤ by GitHub — view raw



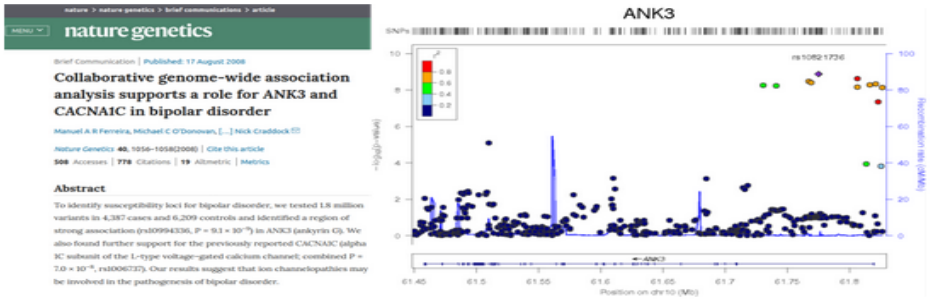

```python
import pandas as pd
from collections import Counter

scores_df = pd.DataFrame({'Base': range(len(perm_scores)),'Score': perm_scores})
informative_indices = list(scores_df[scores_df['Score'] > 0.01].index)
reverse_word_map = dict(map(reversed, tokenizer.word_index.items()))
def sequence_to_text(list_of_indices):
    return [reverse_word_map.get(letter) for letter in list_of_indices]
my_texts = list(map(sequence_to_text, X_test[informative_indices, :]))

fig = plt.figure(figsize = (20, 15))
D = dict(Counter([item.upper() for sublist in my_texts
                  for item in sublist]).most_common(20))
plt.bar(range(len(D)), list(D.values()), align = 'center')
plt.title('Most Predictive K-mers', fontsize = 20)
plt.ylabel("Counts", fontsize = 20); plt.xticks(rotation = 90)
plt.xticks(range(len(D)), list(D.keys()), fontsize = 20)
```

kmer_importance.py hosted with ❤ by GitHub — view raw

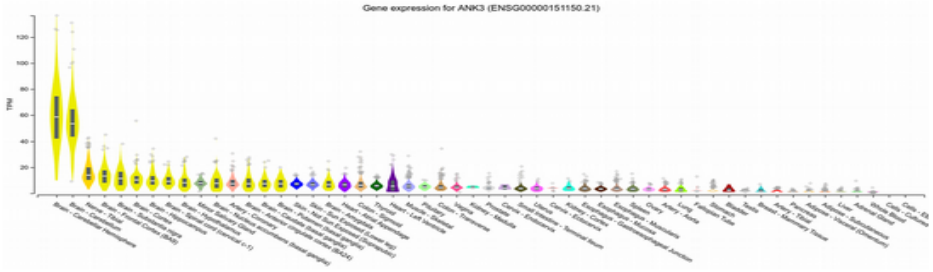


Predict Neanderthal Genes with LSTM

	Gene	Gene_Symbol	Predict	Prob
7360	chr12:120884241-120901556	GATC	1	1.000000
23845	chr6:73844526-73853237	KCNQ5-AS1	1	1.000000
20238	chr4:40194587-40246384	RHOH	1	1.000000
2022	chr1:174769035-174964445	RABGAP1L	1	1.000000
2024	chr1:174904084-174923398	LOC101928696	1	1.000000
8518	chr14:75230069-75304013	YLPM1	1	1.000000
2068	chr1:180199433-180244188	LHX4	1	1.000000
8408	chr14:61176256-61190852	SIX4	1	1.000000
2107	chr1:186369704-186370587	OCLM	1	1.000000
2114	chr1:187412760-187446354	LINC01037	1	1.000000
8219	chr14:24641234-24649463	REC8	1	1.000000
8198	chr14:24422944-24438488	DHRS4	1	1.000000
8109	chr14:20779527-20801471	CCNB1IP1	1	1.000000
2204	chr1:202955580-202976393	LOC100506747	1	1.000000
8033	chr13:111766159-111768025	ARHGEF7-AS2	1	1.000000
7926	chr13:85937738-86118797	LINC00351	1	1.000000
26185	chr7:33019086-33046543	FKBP9	1	1.000000
7701	chr13:39106137-39260812	LINC00437	1	1.000000
2418	chr1:226335704-226342678	ACBD3-AS1	1	1.000000
7465	chr12:125396191-125399587	UBC	1	1.000000
29638	chrUn_gl000211:48503-93165	FLJ43315	1	1.000000
18818	chr3:99979661-100044096	TBC1D23	1	1.000000
17116	chr20:45947246-45949498	LOC100131496	1	1.000000
8943	chr15:45406523-45410301	DUOXA2	1	1.000000
8994	chr15:55647421-55700708	CCPG1	1	1.000000
31193	chrX:154718673-154842622	TMLHE	1	1.000000

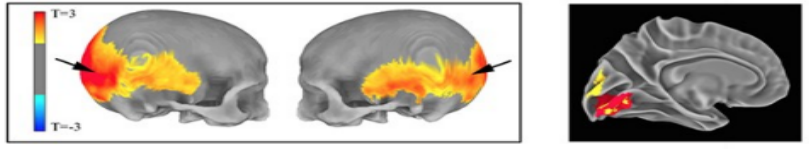


ANK3 gene predicted to be of Neanderthal origin is linked to Bipolar Disorder



ANK3 is a human brain-specific gene based on GTEX human tissue expression data

In addition, some studies suggested that parts of human brains implicated in **mental disorders** may be shaped by genetic inheritance from Neanderthals.



Human brain harbors "residual echo" of Neanderthal genes, image source

Session summary

Take home messages of the session:

1) One-hot-encoding is essential to train CNN on DNA

2) Deep Learning powerful for functional element detection

3) Genomics is a potential Big Data with correct setup

4) RNNs do not outperform CNNs for genomic applications

5) DNA is a text, which is suitable for NLP applications

National Bioinformatics Infrastructure Sweden (NBIS)

Knut och Alice Wallenbergs Stiftelse

Vetenskapsrådet

Lunds Universitet