Machine Learning using Python : Implementation of HMM classifier and CNN Models

Neelotpal Chakraborty

Department of Computer Science and Engineering

Jadavpur University

Hidden Markov Model (HMM)

Install hmmlearn:

```
C:\Users\admin>py -m pip install hmmlearn
Collecting hmmlearn
  Downloading hmmlearn-0.2.6-cp39-cp39-win_amd64.whl (118 kB)
     |                                                | 118 kB 139 kB/s
Requirement already satisfied: scikit-learn>=0.16 in c:\users\ad
ages (from hmmlearn) (0.24.2)
Requirement already satisfied: numpy>=1.10 in c:\users\admin\app
rom hmmlearn) (1.19.5)
Requirement already satisfied: scipy>=0.19 in c:\users\admin\app
rom hmmlearn) (1.7.1)
Requirement already satisfied: joblib>=0.11 in c:\users\admin\ap
from scikit-learn>=0.16->hmmlearn) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\
ckages (from scikit-learn>=0.16->hmmlearn) (2.2.0)
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.2.6
```

Hidden Markov Model (HMM)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Dataset Preparation
dataset = pd.read_csv("E:/JU ML LAB/ML using Python/Datasets/loan_data.csv")
X = dataset.drop(['credit.policy','purpose'], axis=1)
y = dataset['credit.policy']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)


from hmmlearn import hmm
classifier = hmm.GaussianHMM(n_components=2, covariance_type="full", n_iter=1000)

classifier.fit(X_train)

y_pred = classifier.predict(X_test)

# Evaluation of Classifier Performance

from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------------")
print("--------------------------------------------------------")

print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

Output:

```
Confusion Matrix:
[[  79  292]
 [ 259 1286]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.23      0.21      0.22       371
           1       0.81      0.83      0.82      1545

    accuracy                           0.71      1916
   macro avg       0.52      0.52      0.52      1916
weighted avg       0.70      0.71      0.71      1916
```

Hidden Markov Model (HMM)

Note:

```
classifier.fit(X_train)

y_pred = classifier.predict(X_test)

# Evaluation of Classifier Performance

from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("--------------------------------------------------------")
print("--------------------------------------------------------")

print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
```

Hidden Markov Model (HMM) Versions & Parameters

_BaseHMM

```
class hmmlearn.base._BaseHMM(n_components=1, startprob_prior=1.0, transmat_prior=1.0,
algorithm='viterbi', random_state=None, n_iter=10, tol=0.01, verbose=False,
params='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ',
init_params='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
```

Base class for Hidden Markov Models.

This class allows for easy evaluation of, sampling from, and maximum a posteriori estimation of the parameters of a HMM.

Hidden Markov Model (HMM) Versions & Parameters

Parameters:

- **n_components** (*int*) – Number of states in the model.
- **startprob_prior** (*array, shape (n_components,), optional*) – Parameters of the Dirichlet prior distribution for `startprob_`.
- **transmat_prior** (*array, shape (n_components, n_components), optional*) – Parameters of the Dirichlet prior distribution for each row of the transition probabilities `transmat_`.
- **algorithm** (*{"viterbi", "map"}, optional*) – Decoder algorithm.
- **random_state** (*RandomState or an int seed, optional*) – A random number generator instance.
- **n_iter** (*int, optional*) – Maximum number of iterations to perform.
- **tol** (*float, optional*) – Convergence threshold. EM will stop if the gain in log-likelihood is below this value.
- **verbose** (*bool, optional*) – Whether per-iteration convergence reports are printed to `sys.stderr`. Convergence can also be diagnosed using the `monitor_` attribute.
- **params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, and other characters for subclass-specific emission parameters. Defaults to all parameters.
- **init_params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, and other characters for subclass-specific emission parameters. Defaults to all parameters.

Hidden Markov Model (HMM) Versions & Parameters

GaussianHMM

```
class hmmlearn.hmm.GaussianHMM(n_components=1, covariance_type='diag',
min_covar=0.001, startprob_prior=1.0, transmat_prior=1.0, means_prior=0,
means_weight=0, covars_prior=0.01, covars_weight=1, algorithm='viterbi',
random_state=None, n_iter=10, tol=0.01, verbose=False, params='stmc',
init_params='stmc')
```

Hidden Markov Model with Gaussian emissions.

Hidden Markov Model (HMM) Versions & Parameters

Parameters:

- **n_components** (*int*) – Number of states.
- **covariance_type** (*{"sperical", "diag", "full", "tied"}, optional*) –
  The type of covariance parameters to use:
  - "spherical" — each state uses a single variance value that applies to all features (default).
  - "diag" — each state uses a diagonal covariance matrix.
  - "full" — each state uses a full (i.e. unrestricted) covariance matrix.
  - "tied" — all states use **the same** full covariance matrix.
- **min_covar** (*float, optional*) – Floor on the diagonal of the covariance matrix to prevent overfitting. Defaults to 1e-3.
- **startprob_prior** (*array, shape (n_components,), optional*) – Parameters of the Dirichlet prior distribution for `startprob_`.
- **transmat_prior** (*array, shape (n_components, n_components), optional*) –
  Parameters of the Dirichlet prior distribution for each row of the transition probabilities `transmat_`.
- **means_prior** (*array, shape (n_components,), optional*) – Mean and precision of the Normal prior distribtion for `means_`.
- **means_weight** (*array, shape (n_components,), optional*) – Mean and precision of the Normal prior distribtion for `means_`.
- **covars_prior** (*array, shape (n_components,), optional*) –
  Parameters of the prior distribution for the covariance matrix `covars_`.
  If `covariance_type` is "spherical" or "diag" the prior is the inverse gamma distribution, otherwise — the inverse Wishart distribution.

Hidden Markov Model (HMM) Versions & Parameters

- **covars_weight** (*array, shape (n_components,), optional*) – Parameters of the prior distribution for the covariance matrix `covars_`. If `covariance_type` is "spherical" or "diag" the prior is the inverse gamma distribution, otherwise — the inverse Wishart distribution.
- **algorithm** (*{"viterbi", "map"}, optional*) – Decoder algorithm.
- **random_state** (*RandomState or an int seed, optional*) – A random number generator instance.
- **n_iter** (*int, optional*) – Maximum number of iterations to perform.
- **tol** (*float, optional*) – Convergence threshold. EM will stop if the gain in log-likelihood is below this value.
- **verbose** (*bool, optional*) – Whether per-iteration convergence reports are printed to `sys.stderr`. Convergence can also be diagnosed using the `monitor_` attribute.
- **params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, 'm' for means, and 'c' for covars. Defaults to all parameters.
- **init_params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, 'm' for means, and 'c' for covars. Defaults to all parameters.

Hidden Markov Model (HMM) Versions & Parameters

GMMHMM

```
class hmmlearn.hmm.GMMHMM(n_components=1, n_mix=1, min_covar=0.001,
startprob_prior=1.0, transmat_prior=1.0, weights_prior=1.0, means_prior=0.0,
means_weight=0.0, covars_prior=None, covars_weight=None, algorithm='viterbi',
covariance_type='diag', random_state=None, n_iter=10, tol=0.01, verbose=False,
params='stmcw', init_params='stmcw')
```

Hidden Markov Model with Gaussian mixture emissions.

Hidden Markov Model (HMM) Versions & Parameters

Parameters:

- **n_components** (*int*) – Number of states in the model.
- **n_mix** (*int*) – Number of states in the GMM.
- **covariance_type** (*{"sperical", "diag", "full", "tied"}, optional*) –
  The type of covariance parameters to use:
  - "spherical" — each state uses a single variance value that applies to all features.
  - "diag" — each state uses a diagonal covariance matrix (default).
  - "full" — each state uses a full (i.e. unrestricted) covariance matrix.
  - "tied" — all mixture components of each state use **the same** full covariance matrix (note that this is not the same as for `GaussianHMM`).
- **min_covar** (*float, optional*) – Floor on the diagonal of the covariance matrix to prevent overfitting. Defaults to 1e-3.
- **startprob_prior** (*array, shape (n_components,), optional*) – Parameters of the Dirichlet prior distribution for `startprob_`.
- **transmat_prior** (*array, shape (n_components, n_components), optional*) – Parameters of the Dirichlet prior distribution for each row of the transition probabilities `transmat_`.
- **weights_prior** (*array, shape (n_mix,), optional*) – Parameters of the Dirichlet prior distribution for `weights_`.
- **means_prior** (*array, shape (n_mix,), optional*) – Mean and precision of the Normal prior distribtion for `means_`.
- **means_weight** (*array, shape (n_mix,), optional*) – Mean and precision of the Normal prior distribtion for `means_`.

Hidden Markov Model (HMM) Versions & Parameters

- **covars_prior** (*array, shape (n_mix,), optional*) –
  Parameters of the prior distribution for the covariance matrix `covars_`.
  If `covariance_type` is "spherical" or "diag" the prior is the inverse gamma distribution, otherwise — the inverse Wishart distribution.
- **covars_weight** (*array, shape (n_mix,), optional*) –
  Parameters of the prior distribution for the covariance matrix `covars_`.
  If `covariance_type` is "spherical" or "diag" the prior is the inverse gamma distribution, otherwise — the inverse Wishart distribution.
- **algorithm** (*{"viterbi", "map"}, optional*) – Decoder algorithm.
- **random_state** (*RandomState or an int seed, optional*) – A random number generator instance.
- **n_iter** (*int, optional*) – Maximum number of iterations to perform.
- **tol** (*float, optional*) – Convergence threshold. EM will stop if the gain in log-likelihood is below this value.
- **verbose** (*bool, optional*) – Whether per-iteration convergence reports are printed to `sys.stderr`. Convergence can also be diagnosed using the `monitor_` attribute.
- **params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, 'm' for means, 'c' for covars, and 'w' for GMM mixing weights. Defaults to all parameters.
- **init_params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, 'm' for means, 'c' for covars, and 'w' for GMM mixing weights. Defaults to all parameters.

Hidden Markov Model (HMM) Versions & Parameters

MultinomialHMM

```
class hmmlearn.hmm.MultinomialHMM(n_components=1, startprob_prior=1.0,
transmat_prior=1.0, algorithm='viterbi', random_state=None, n_iter=10, tol=0.01,
verbose=False, params='ste', init_params='ste')
```

Hidden Markov Model with multinomial (discrete) emissions.

Hidden Markov Model (HMM) Versions & Parameters

Parameters:

- **n_components** (*int*) – Number of states.
- **startprob_prior** (*array, shape (n_components,), optional*) – Parameters of the Dirichlet prior distribution for `startprob_`.
- **transmat_prior** (*array, shape (n_components, n_components), optional*) – Parameters of the Dirichlet prior distribution for each row of the transition probabilities `transmat_`.
- **algorithm** (*{"viterbi", "map"}, optional*) – Decoder algorithm.
- **random_state** (*RandomState or an int seed, optional*) – A random number generator instance.
- **n_iter** (*int, optional*) – Maximum number of iterations to perform.
- **tol** (*float, optional*) – Convergence threshold. EM will stop if the gain in log-likelihood is below this value.
- **verbose** (*bool, optional*) – Whether per-iteration convergence reports are printed to `sys.stderr`. Convergence can also be diagnosed using the `monitor_` attribute.
- **params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, and 'e' for emissionprob. Defaults to all parameters.
- **init_params** (*string, optional*) – The parameters that get updated during (`params`) or initialized before (`init_params`) the training. Can contain any combination of 's' for startprob, 't' for transmat, and 'e' for emissionprob. Defaults to all parameters.

Links for more details on HMM function

https://hmmlearn.readthedocs.io/en/stable/tutorial.html#available-models

https://hmmlearn.readthedocs.io/en/stable/api.html#hmmlearn.hmm.GMMHMM

Classification: Convolutional Neural Network (CNN)- Using Google Colab

CNNdemo.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf


# Dataset Preparation

from tensorflow.keras import datasets, layers, models

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

Classification: Convolutional Neural Network (CNN)- Using Google Colab

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.summary()
```

Classification: Convolutional Neural Network (CNN)- Using Google Colab

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
#plt.show()

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

Classification: Convolutional Neural Network (CNN)- Using Google Colab

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

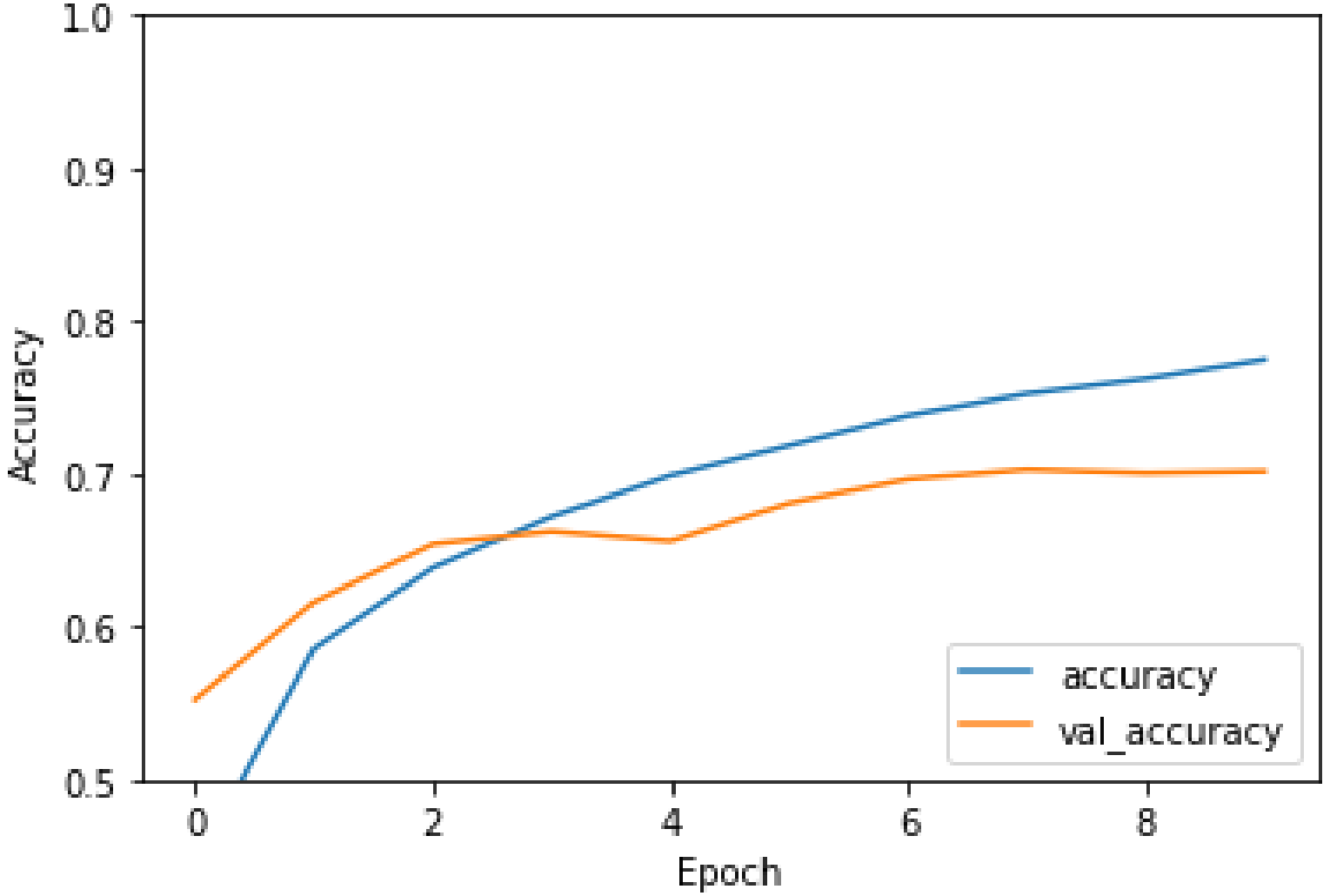
Classification: Convolutional Neural Network (CNN)- Using Google Colab

```
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Classification: Convolutional Neural Network (CNN)- Using Google Colab

```
_________________________________________________________________
Epoch 1/10
1563/1563 [==============================] - 68s 43ms/step - loss: 1.5209 - accuracy: 0.4446 - val_loss: 1.2589 - val_accuracy: 0.5522
Epoch 2/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.1690 - accuracy: 0.5856 - val_loss: 1.0857 - val_accuracy: 0.6161
Epoch 3/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.0263 - accuracy: 0.6385 - val_loss: 0.9984 - val_accuracy: 0.6540
Epoch 4/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.9300 - accuracy: 0.6721 - val_loss: 0.9657 - val_accuracy: 0.6623
Epoch 5/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.8569 - accuracy: 0.6988 - val_loss: 0.9834 - val_accuracy: 0.6560
Epoch 6/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.7997 - accuracy: 0.7186 - val_loss: 0.9407 - val_accuracy: 0.6809
Epoch 7/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.7506 - accuracy: 0.7382 - val_loss: 0.8955 - val_accuracy: 0.6964
Epoch 8/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.7082 - accuracy: 0.7524 - val_loss: 0.8806 - val_accuracy: 0.7026
Epoch 9/10
1563/1563 [==============================] - 67s 43ms/step - loss: 0.6714 - accuracy: 0.7621 - val_loss: 0.8798 - val_accuracy: 0.7003
Epoch 10/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.6383 - accuracy: 0.7746 - val_loss: 0.8718 - val_accuracy: 0.7015
313/313 - 3s - loss: 0.8718 - accuracy: 0.7015
```

Classification: Convolutional Neural Network (CNN)- Using Google Colab



Some Standards Datasets- A Brief Overview

CIFAR-10: https://www.cs.toronto.edu/~kriz/cifar.html

MNIST: http://yann.lecun.com/exdb/mnist/

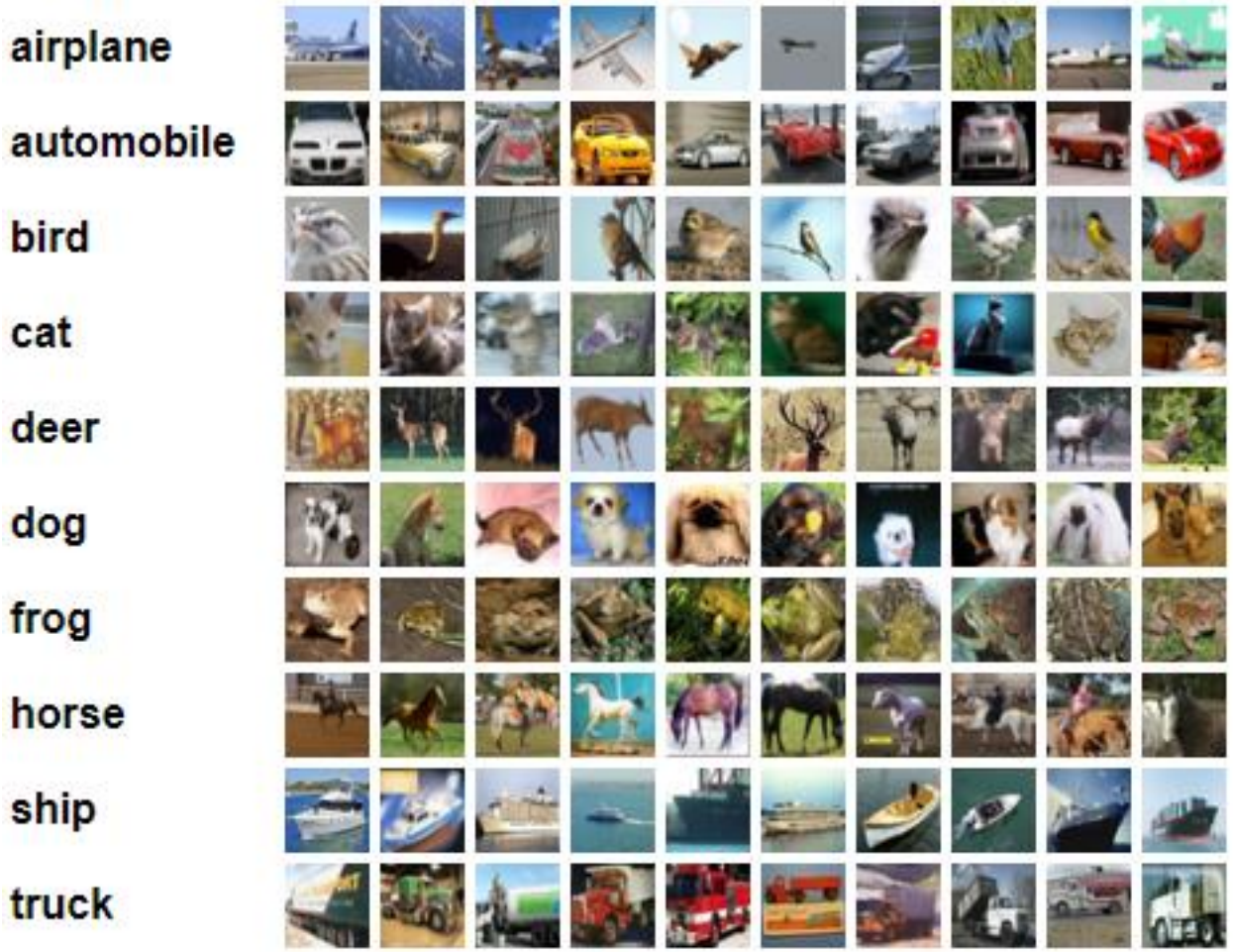
SAVEE: http://kahlan.eps.surrey.ac.uk/savee/Download.html

EmoDB: http://www.emodb.bilderbar.info/navi.html

Some Standards Datasets- A Brief Overview

The CIFAR-10 dataset:

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
- There are 50000 training images and 10000 test images.
- The dataset is divided into five training batches and one test batch, each with 10000 images.
- The test batch contains exactly 1000 randomly-selected images from each class.
- The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another.
- Between them, the training batches contain exactly 5000 images from each class.



Some Standards Datasets- A Brief Overview

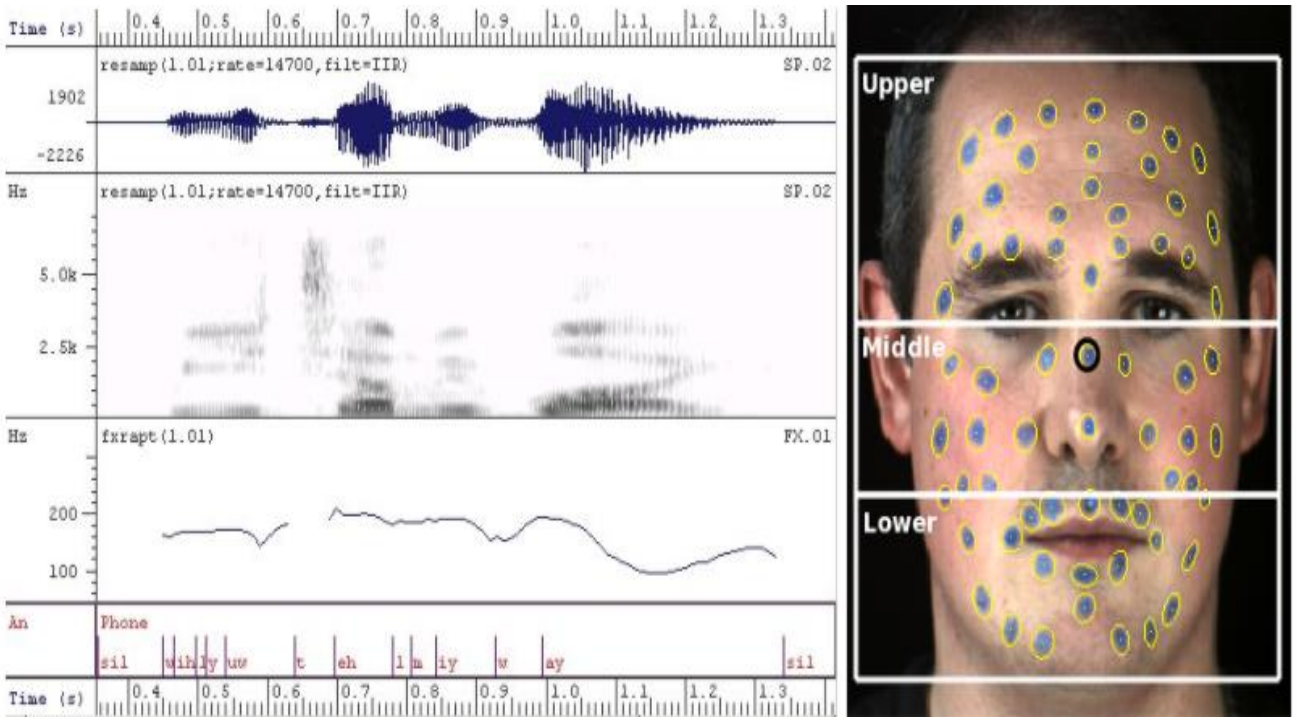
THE MNIST DATABASE of handwritten digits:

- It has a training set of 60,000 examples, and a test set of 10,000 examples.
- It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

Some Standards Datasets- A Brief Overview

Surrey Audio-Visual Expressed Emotion (SAVEE) Database:

- The SAVEE database was recorded from four native English male speakers (identified as DC, JE, JK, KL), postgraduate students and researchers at the University of Surrey aged from 27 to 31 years.
- Emotion has been described psychologically in discrete categories: anger, disgust, fear, happiness, sadness and surprise.
- Example:
    1. **Common:** She had your dark suit in greasy wash water all year.
    2. **Anger:** Who authorized the unlimited expense account?
    3. **Disgust:** Please take this dirty table cloth to the cleaners for me.
    4. **Fear:** Call an ambulance for medical assistance.
    5. **Happiness:** Those musicians harmonize marvelously.
    6. **Sadness:** The prospect of cutting back spending is an unpleasant one for any governor.
    7. **Surprise:** The carpet cleaners shampooed our oriental rug.
    8. **Neutral:** The best way to learn is to solve extra problems.



Audio feature extraction with Speech Filing System software (left), and visual data (right) with tracked marker locations. Marker on the bridge of nose (encircled in black) was taken as a reference.

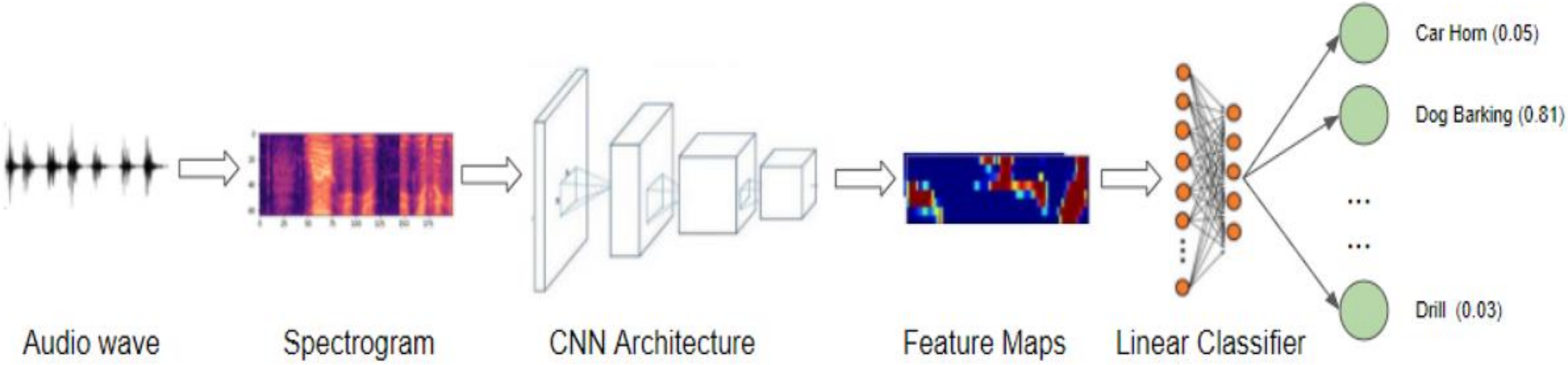
Some Standards Datasets- A Brief Overview

Berlin Database of Emotional Speech **(Berlin Database of Emotional Speech):**

- The EMODB database is the freely available German emotional database. The database is created by the Institute of Communication Science, Technical University, Berlin, Germany.
- Ten professional speakers (five males and five females) participated in data recording.
- The database contains a total of 535 utterances. The EMODB database comprises of seven emotions: 1) anger; 2) boredom; 3) anxiety; 4) happiness; 5) sadness; 6) disgust; and 7) neutral.
- The data was recorded at a 48-kHz sampling rate and then down-sampled to 16-kHz.

Code of emotions:			
Letter	Emotion (English)	Letter	Emotion (German)
A	anger	W	Ärger (Wut)
B	boredom	L	Langeweile
D	disgust	E	Ekel
F	anxiety/fear	A	Angst
H	happiness	F	Freude
S	sadness	T	Trauer
N = neutral version			

Working with Audio Signal Datasets- A Brief Overview



1. Convert each audio file to an image/spectrogram in both train & test sets.

2. Train & test a CNN model on these images

Link for audio signal classification: https://www.kaggle.com/timolee/audio-data-conversion-to-images-eda