Mathematical foundations of Machine Learning 2024 – lesson 1

Shai Dekel

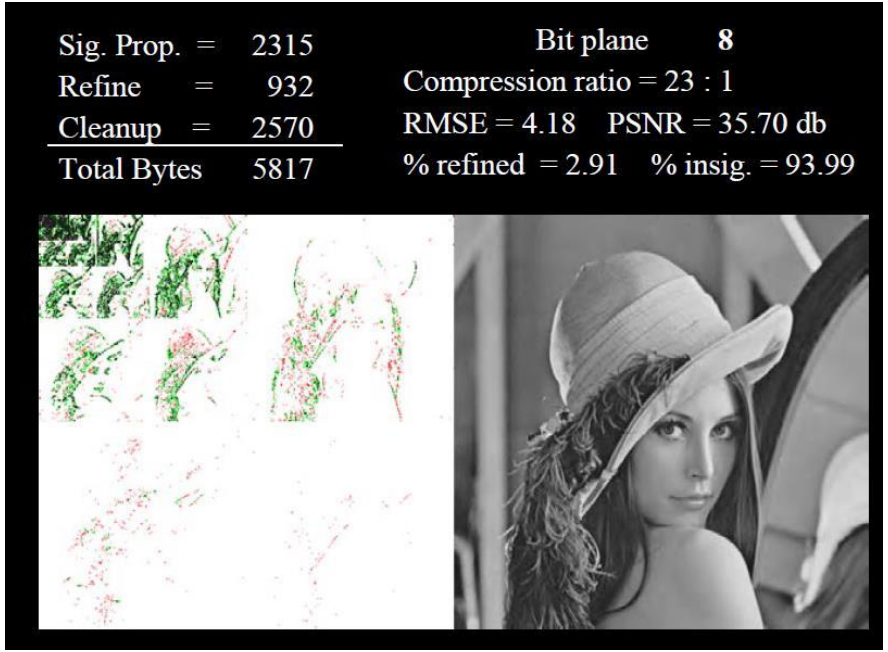
TEL AVIV UNIVERSITY

Our goal is to provide a holistic mathematical foundation for Artificial Intelligence (AI) which includes classic Machine Learning (ML) and Deep Learning (DL) through:

Function space theory & Approximation Theory

Mathematical foundation of signal processing

- Dataset is $\left\{(i,j), f(i,j)\right\}_{i,j=1}^{512}$
- **The analysis of an image is based on the 'geometry' of the data: clusters of pixels and edges.**
- We may use wavelets to decompose the pixel data. Wavelets represent multi-scale edges.
- We can characterize the performance of wavelet image compression by Besov smoothness of image (as a function).
- Approximation theory: characterization of the performance of models/algorithms using function space (weak-type) smoothness.



kaggle.com/datasets

kaggle

Search

Sign In

Register

Create

Home

Competitions

Datasets

Code

Discussions

Courses

More

Datasets

Explore, analyze, and share quality data. Learn more about data types, creating, and collaborating.

+ New Dataset

Search datasets

Filters

Computer Science | Education | Classification | Computer Vision | NLP | Data Visualization

Trending Datasets

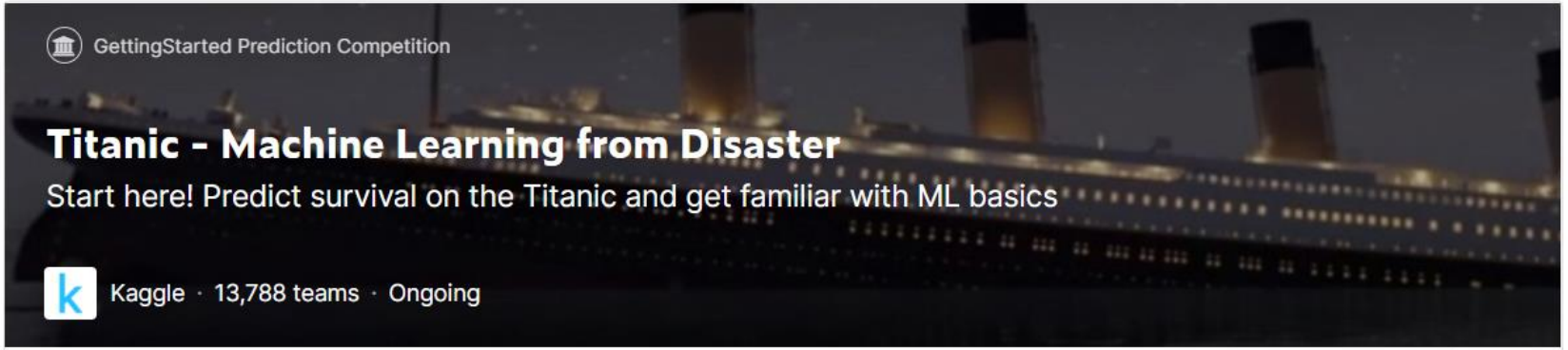
See All

10,000_Amazon_Products_D

Atp Tennis Data (2008- Feb

Reddit: r/Sadhguru Hot

Quordle JavaScript Source



Overview

👏🚂 Ahoy, welcome to Kaggle! You're in the right place.

This is the legendary Titanic ML competition – the best, first challenge for you to dive into ML competitions and familiarize yourself with how the Kaggle platform works.

The competition is simple: use machine learning to create a model that predicts which passengers survived the Titanic shipwreck.

Read on or watch the video below to explore more details. Once you're ready to start competing, click on the "Join Competition button to create an account and gain access to the competition data. Then check out Alexis Cook's Titanic Tutorial that walks you through step by step how to make your first submission!

```
]: #os.listdir()
df = pandas.read_csv('C:\\Users\\user\\Google Drive\\academic\\code\\data\\Titanic\\train_data.csv')
df.head()# Analysing the data set and its columns
```

]:

	Unnamed: 0	PassengerId	Survived	Gender	Age	Fare	Pclass_1	Pclass_2	Pclass_3	Family_size	Title_1	Title_2	Title_3	Title_4	Emb_1	Emb_2	Emb_3
0	0	1	0	1	0.2750	0.014151	0	0	1	0.1	1	0	0	0	0	0	1
1	1	2	1	0	0.4750	0.139136	1	0	0	0.1	1	0	0	0	1	0	0
2	2	3	1	0	0.3250	0.015469	0	0	1	0.0	0	0	0	1	0	0	1
3	3	4	1	0	0.4375	0.103644	1	0	0	0.1	1	0	0	0	0	0	1
4	4	5	0	1	0.4375	0.015713	0	0	1	0.0	1	0	0	0	0	0	1

unused

Y

X

normalized

Class (category) vector encoding

$$f: [0,1]^{14} \rightarrow \{0,1\}$$

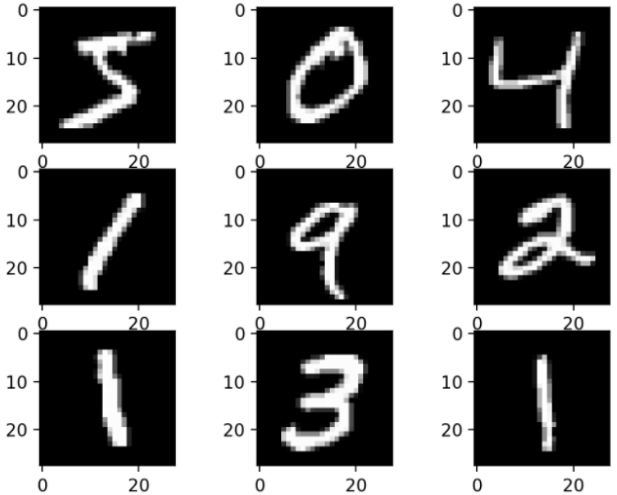
Function space representation of an image dataset

- Assume we have a dataset of $\sqrt{n_0} \times \sqrt{n_0}$ grayscale images with pixels in [0,255].
- We concatenate the pixel values to vectors of size n_0
- We normalize the pixels values to $[0,1]$
- Each image is associated with one of L class labels.
- We map each label to its one-hot-encoding in $\mathbb{R}^L$ $(0,0,\dots,1,0,\dots 0)$
- Thus, each image is now a sample of a function

$$f_0 : [0,1]^{n_0} \to \mathbb{R}^L$$

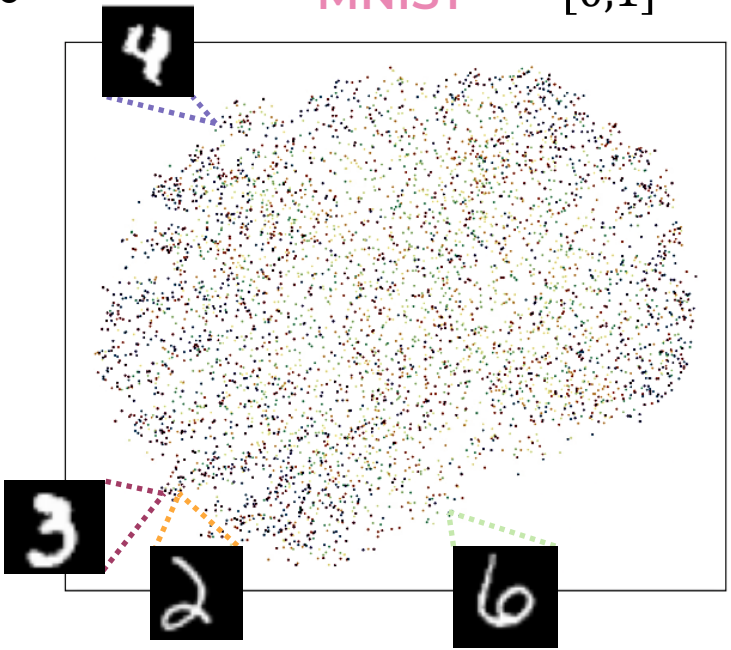
MNIST

Hand written digits

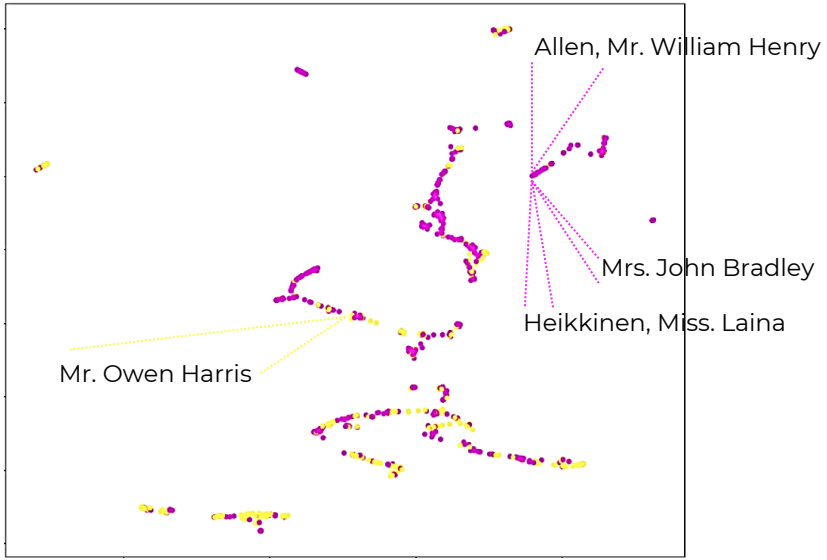


Machine Learning vs. Deep Learning

- When do we want to use classic ML vs. DL for classification?
- Is there ‘geometry’ in the feature space?
- MNIST $[0,1]^{784}$

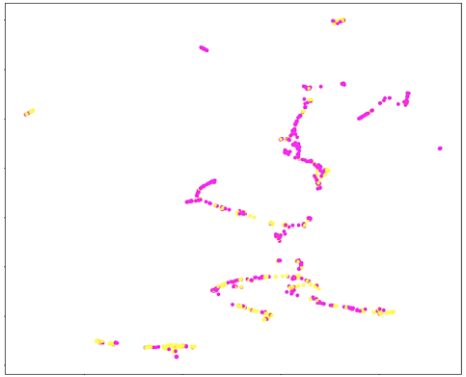


TITANIC $[0,1]^{14}$



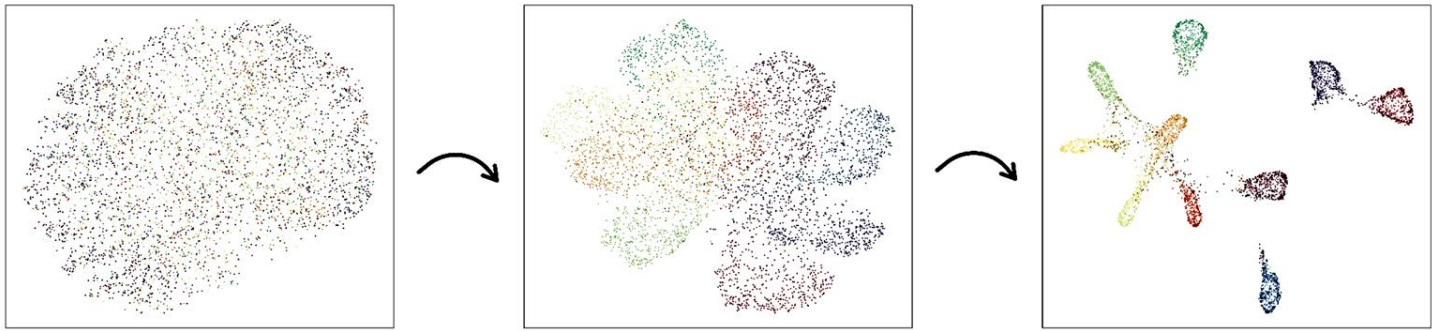
McInnes, L., Healy, J., and Melville, J., “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction”, arXiv e-prints, 2018.

ML vs. DL



- Good separability in input feature space → ML
- All successful Machine Learning algorithms look for this geometry:
  - Support Vector Machines, Random Forest, Gradient Boosting, etc.
- **If not, can we transform to a better feature space through feature engineering/deep learning (CNN, Resnets, Transformers etc)?**

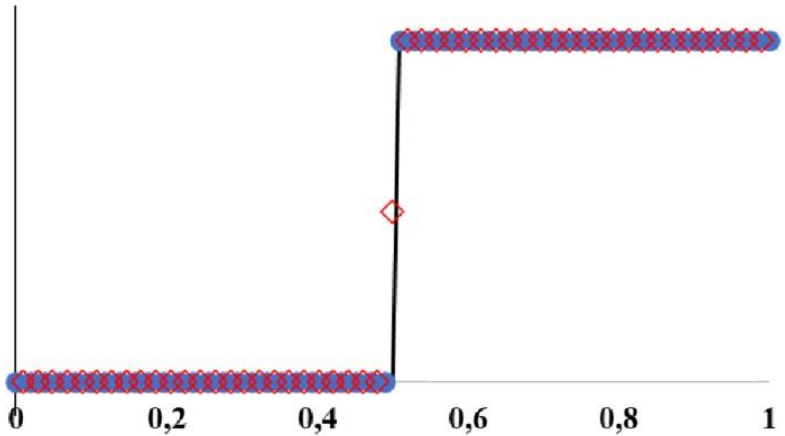
MNIST DATASET



[1] UMAP of trained ConvNet on MNIST Dataset - Ben-Shaul, I. and Dekel, S., “Sparsity-Probe: Analysis tool for Deep Learning Models”, arXiv e-prints, 2021.
[2] F. Chollet, Deep Learning with Python, Manning, Nov. 2017.

Smoothness? Yes! Weak-type smoothness using Besov spaces

$$f \in B_{\tau}^{\alpha}, \quad \forall \alpha > 0$$



Training better models using this theory! Vision… NLP…

	IT		EOT	
Dataset	Vanilla	SVSL	Vanilla	SVSL
MNIST	**99.37**	**99.36**	99.61	**99.69**
Fashion MNIST	91.78	**93.13**	93.82	**93.88**
STL10	53.41	**55.95**	54.11	**56.65**
CIFAR10	**80.64**	80.56	80.96	**81.19**
CIFAR100	52.77	**53.28**	53.31	**54.29**
CoLA	51.59	**52.91**	53.46	**55.54**
RTE	**58.84**	58.12	55.23	**59.57**
MRPC	70.83	**74.26**	74.26	**75.25**
SST-2	87.96	**88.42**	88.42	**88.76**

*I. Ben-Shaul & S. Dekel, Proceedings of machine learning, 2022.

Current state of the art in AI: Transformers

DeepMind

19 July 2022

Formal Algorithms for Transformers

Mary Phuong[1] and Marcus Hutter[1]
[1]DeepMind

This document aims to be a self-contained, mathematically precise overview of transformer architectures and algorithms (*not* results). It covers what transformers are, how they are trained, what they are used for, their key architectural components, and a preview of the most prominent models. The reader is assumed to be familiar with basic ML terminology and simpler neural network architectures such as MLPs.

Keywords: formal algorithms, pseudocode, transformers, attention, encoder, decoder, BERT, GPT, Gopher, tokenization, training, inference.

GPT-4 Turbo and GPT-4

GPT-4 is a large multimodal model (accepting text or image inputs and outputting text) that can solve difficult problems with greater accuracy than any of our previous models, thanks to its broader general knowledge and advanced reasoning capabilities. GPT-4 is available in the OpenAI API to paying customers. Like `gpt-3.5-turbo`, GPT-4 is optimized for chat but works well for traditional completions tasks using the Chat Completions API. Learn how to use GPT-4 in our text generation guide.

MODEL	DESCRIPTION	CONTEXT WINDOW	TRAINING DATA
`gpt-4-turbo`	New **GPT-4 Turbo with Vision** The latest GPT-4 Turbo model with vision capabilities. Vision requests can now use JSON mode and function calling. Currently points to `gpt-4-turbo-2024-04-09`.	128,000 tokens	Up to Dec 2023
`gpt-4-turbo-2024-04-09`	GPT-4 Turbo with Vision model. Vision requests can now use JSON mode and function calling. `gpt-4-turbo` currently points to this version.	128,000 tokens	Up to Dec 2023
`gpt-4-turbo-preview`	GPT-4 Turbo preview model. Currently points to `gpt-4-0125-preview`.	128,000 tokens	Up to Dec 2023
`gpt-4-0125-preview`	GPT-4 Turbo preview model intended to reduce cases of “laziness” where the model doesn’t complete a task. Returns a maximum of 4,096 output tokens. Learn more.	128,000 tokens	Up to Dec 2023

Generative visual models

DALL·E

DALL·E is a AI system that can create realistic images and art from a description in natural language. DALL·E 3 currently supports the ability, given a prompt, to create a new image with a specific size. DALL·E 2 also support the ability to edit an existing image, or create variations of a user provided image.

DALL·E 3 is available through our Images API along with DALL·E 2. You can try DALL·E 3 through ChatGPT Plus.

MODEL	DESCRIPTION
`dall-e-3`	New **DALL·E 3** The latest DALL·E model released in Nov 2023. Learn more.

Stable Diffusion 3

Stable Diffusion 3, our most advanced image model yet, features the latest in text-to-image technology with greatly improved performance in multi-subject prompts, image quality, and spelling abilities.

The model is available via API today and we are continuously working to improve the model in advance of its open release. To learn more about deployment options please contact us.

Here we go...let’s start at the beginning ...

Binary Classification

We build a predictive model for the problem based on training data.

Example for problem: (x_i, y_i), $x_i \in \mathbb{R}^n$ - vector of patient data,

$y_i \in \{0,1\}$ - response variable = {no risk, potential health issue}

Option I linear regression (typically used for case where $y_i \in \mathbb{R}$). We train for unknown parameters

$\theta = (\beta, \beta_0)$, weights $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{R}^n$ and bias $\beta_0 \in \mathbb{R}$

$$\hat{y}_i = \langle \beta, x_i \rangle + \beta_0 = \sum_{j=1}^{n} \beta_j x_{i,j} + \beta_0 .$$

Loss Function - We solve using the training data

$$\mathrm{Loss}\left(Y \mid X, \tilde{\theta}\right) := \frac{1}{\#I} \sum_{i \in I} \left(\sum_{j=1}^{n} \tilde{\beta}_j x_{i,j} + \tilde{\beta}_0 - y_i \right)^2 ,$$

$$\theta = \arg\min_{\tilde{\theta}} \mathrm{Loss}\left(Y \mid X, \tilde{\theta}\right)$$

Solution is by simple differentiation to find the minima. We get a linear system of dimension $n+1$. For $1 \le k \le n$,

$$\frac{\partial}{\partial \beta_k} \sum_i \left(\sum_{j=1}^{n} \beta_j x_{i,j} + \beta_0 - y_i \right)^2 = 2 \sum_i \left(\sum_{j=1}^{n} \beta_j x_{i,j} + \beta_0 - y_i \right) x_{i,k} = 0 \Leftrightarrow$$

$$\sum_i \left(\sum_{j=1}^{n} \beta_j x_{i,j} x_{i,k} + \beta_0 x_{i,k} - y_i x_{i,k} \right) = 0 \Leftrightarrow$$

$$\sum_{j=1}^{n} \left(\sum_i x_{i,j} x_{i,k} \right) \beta_j + \left(\sum_i x_{i,k} \right) \beta_0 = \sum_i y_i x_{i,k}$$

$$\frac{\partial}{\partial \beta_0} \sum_i \left(\sum_{j=1}^{n} \beta_j x_{i,j} + \beta_0 - y_i \right)^2 = 2 \sum_i \left(\sum_{j=1}^{n} \beta_j x_{i,j} + \beta_0 - y_i \right) = 0 \Leftrightarrow$$

$$\sum_{j=1}^{n} \left(\sum_i x_{i,j} \right) \beta_j + (\#i) \beta_0 = \sum_i y_i$$

Then, for a new incoming data point $x \in \mathbb{R}^n$, we compute (in a regression problem)

$$\hat{y}' = \sum_{j=1}^{n} \beta_j x_j + \beta_0 .$$

For binary classification we perform simple "binning" (two bins in this example)

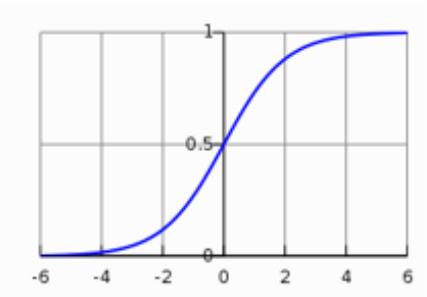
$$\hat{y} := \begin{cases} 0 & \hat{y}' < 0.5 \\ 1 & \hat{y}' \geq 0.5 \end{cases} .$$

From statistical viewpoint - we have not utilized the fact that the problem is "categorical" binary classification.

From approximation theoretical perspective – We have not utilized the fact that we want to approximate a piecewise constant function with a boundary determined by a hyperplane.

The logistic function

$$\sigma(t) := \frac{1}{1+e^{-t}} .$$



$$\sigma(t) \underset{t\to-\infty}{\to} 0,\ \sigma(0) = 0.5,\ \sigma(t) \underset{t\to\infty}{\to} 1$$

We then model with $\theta := \{\beta, \beta_0\} \in \mathbb{R}^{n+1}$

$$\hat{y}' = h_\theta(x) := \frac{1}{1+e^{-(\langle \beta, x\rangle + \beta_0)}} = \frac{1}{1+e^{-\left(\sum_{i=1}^{n} \beta_i x_i + \beta_0\right)}} .$$

Geometry! $\sum_{i=1}^{n} \beta_i x_i + \beta_0 = 0$ is an equation of a (subdividing) hyper-plane.

Statistical Modeling: $\Pr(y \mid x, \theta) = h_\theta(x)^y \left(1 - h_\theta(x)\right)^{1-y}$

So, we want to maximize the likelihood function

$$L(\theta \mid x) = \Pr(Y \mid X, \theta)$$

$$= \prod_{i \in I} \Pr(y_i \mid x_i, \theta)$$

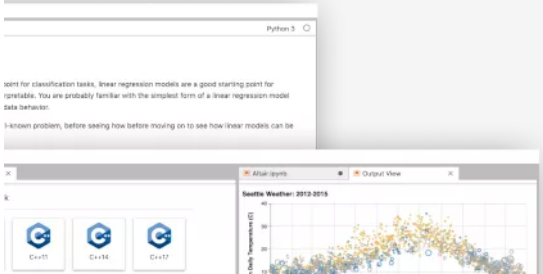
$$= \prod_{i \in I} h_\theta(x_i)^{y_i} \left(1 - h_\theta(x_i)\right)^{1-y_i}$$

One typically normalizes with the size of the data set and minimizes the negative log-likelihood

$$-\frac{1}{\#I} \log L(\theta \mid x) = -\frac{1}{\#I} \log \prod_{i \in I} h_\theta(x_i)^{y_i} \left(1 - h_\theta(x_i)\right)^{1-y_i}$$

$$= -\frac{1}{\#I} \sum_{i \in I} y_i \log h_\theta(x_i) + (1 - y_i) \log\left(1 - h_\theta(x_i)\right)$$

Free software, open standards, and web services for interactive computing across all programming languages



JupyterLab: A Next-Generation Notebook Interface

JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.

Try it in your browser | Install JupyterLab

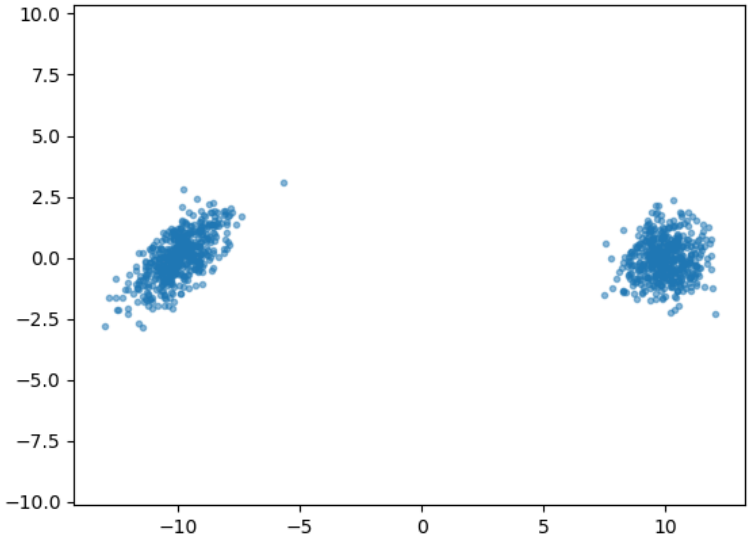
Good clustering

[26]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

[27]:
```
mean1 = [-10, 0]
cov1 = [[1,0.7], [0.7,1]]
pts1 =np.random.default_rng().multivariate_normal(mean1, cov1,size=500)
```

[28]:
```
mean2 = [10, 0]
cov2 = [[0.7,0.1], [0.1,0.7]]
pts2 =np.random.default_rng().multivariate_normal(mean2, cov2,size=500)
```

[29]:
```
plt.plot(np.concatenate((pts1[:,0],pts2[:, 0])), np.concatenate((pts1[:,1],pts2[:, 1])), '.', alpha=0.5)
plt.axis('equal')
plt.show()
```



[5]:
```
x_train = np.concatenate((pts1[0:250],pts2[0:250]))
y_train = np.concatenate((np.zeros(250),np.ones(250)))
x_test = np.concatenate((pts1[250:500],pts2[250:500]))
y_test = y_train
```

[6]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

[6]:
```
▾ LogisticRegression

LogisticRegression()
```

[7]:
```
predict = model.predict(x_test)
print("Accuracy is :")
print(accuracy_score(predict, y_test)*100)
```

```
Accuracy is :
100.0
```

[8]:
```
#Weights of the logistic model / importance of features
print(model.coef_)
```

```
[[ 0.88965294 -0.03212738]]
```

[9]:
```
#Bias ceofficient of the model
print(model.intercept_)
```

```
[-0.01907868]
```

]:
```
#Linear regression model
model=LinearRegression()
model.fit(x_train,y_train)
```

]:
```
▾ LinearRegression

LinearRegression()
```

]:
```
predict = model.predict(x_test)
print("Mean Square Error is :")
print(sum(np.square(predict-y_test))/len(y_test))
```

```
Mean Square Error is :
0.0017137863552906924
```

]:

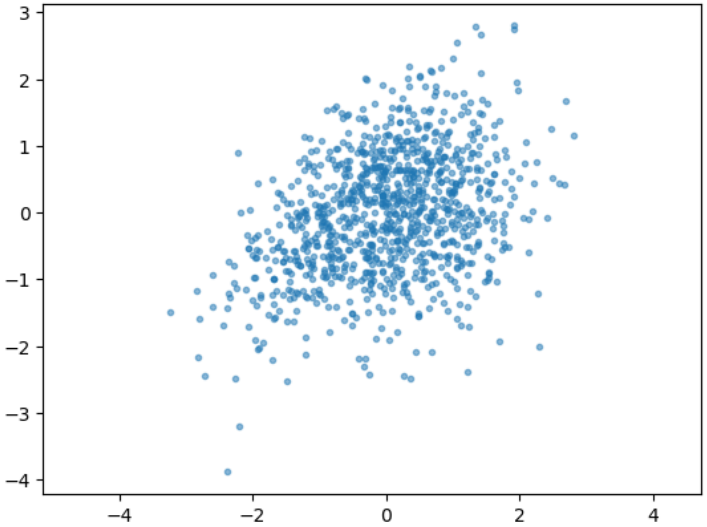
[36]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

[43]:
```
mean1 = [-0.5, 0]
cov1 = [[1,0.7], [0.7,1]]
pts1 =np.random.default_rng().multivariate_normal(mean1, cov1,size=500)
```

[44]:
```
mean2 = [0.5, 0]
cov2 = [[0.7,0.1], [0.1,0.7]]
pts2 =np.random.default_rng().multivariate_normal(mean2, cov2,size=500)
```

Bad clustering

[45]:
```
plt.plot(np.concatenate((pts1[:,0],pts2[:, 0])), np.concatenate((pts1[:,1],pts2[:, 1])), '.', alpha=0.5)
plt.axis('equal')
plt.show()
```



[5]:
```
x_train = np.concatenate((pts1[0:250],pts2[0:250]))
y_train = np.concatenate((np.zeros(250),np.ones(250)))
x_test = np.concatenate((pts1[250:500],pts2[250:500]))
y_test = y_train
```

[6]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

[6]:
```
▾ LogisticRegression

LogisticRegression()
```

[7]:
```
predict = model.predict(x_test)
print("Accuracy is :")
print(accuracy_score(predict, y_test)*100)
```

```
Accuracy is :
74.6
```

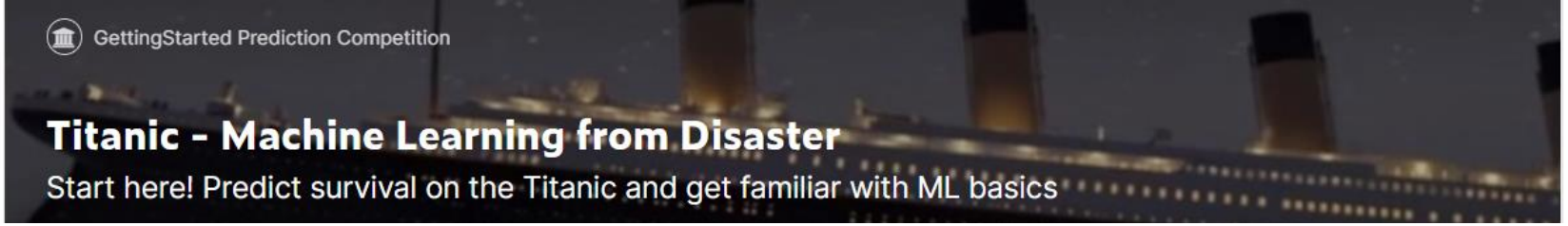
[8]:
```
#Weights of the logistic model / importance of features
print(model.coef_)
```

```
[[ 1.3989976  -0.56374119]]
```

[9]:
```
#Bias ceofficient of the model
print(model.intercept_)
```

```
[0.10592563]
```

[ ]:



[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas
import os

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

[2]:
```
#os.listdir()
df = pandas.read_csv('C:\\Users\\user\\Google Drive\\academic\\code\\data\\Titanic\\train_data.csv')
df.head()# Analysing the data set and its columns
```

[2]:

	Unnamed: 0	PassengerId	Survived	Gender	Age	Fare	Pclass_1	Pclass_2	Pclass_3	Family_size	Title_1	Title_2	Title_3	Title_4	Emb_1	Emb_2	Emb_3
0	0	1	0	1	0.2750	0.014151	0	0	1	0.1	1	0	0	0	0	0	1
1	1	2	1	0	0.4750	0.139136	1	0	0	0.1	1	0	0	0	1	0	0
2	2	3	1	0	0.3250	0.015469	0	0	1	0.0	0	0	0	1	0	0	1
3	3	4	1	0	0.4375	0.103644	1	0	0	0.1	1	0	0	0	0	0	1
4	4	5	0	1	0.4375	0.015713	0	0	1	0.0	1	0	0	0	0	0	1

3]:
```
def get_data(string):
    if string=="train":
        df = pandas.read_csv('C:\\Users\\user\\Google Drive\\academic\\code\\data\\Titanic\\train_data.csv')

    else:
        df=pandas.read_csv('C:\\Users\\user\\Google Drive\\academic\\code\\data\\Titanic\\test_data.csv')

    # splitting to features and target variables
    X = np.asarray(df.loc[:,"Gender":])
    Y = np.asarray(df.loc[:,"Survived"]).reshape(df["Survived"].shape[0],1)
    Y = Y.ravel()

    return X,Y
```

4]:
```
X_train,Y_train = get_data("train")
X_test , Y_test = get_data("test")
```

5]:
```
model=LogisticRegression()
model.fit(X_train,Y_train)
```

5]:
▾ LogisticRegression

LogisticRegression()

[6]:
```
predict = model.predict(X_test)
print("Accuracy is :")
print(accuracy_score(predict, Y_test)*100)
```

```
Accuracy is :
88.0
```

[7]:
```
#Weights of the logistic model / importance of features
print(model.coef_)
```

```
[[-2.91009558 -1.08623477  0.46247683  0.9332186   0.10639119 -1.03832063
  -2.06333792 -0.87767216  0.13053625  1.64417078 -0.89574571  0.20037446
  -0.02006746 -0.25060282]]
```

[8]:

Gender	Age	Fare	Pclass_1	Pclass_2	Pclass_3	Family_size	Title_1	Title_2	Title_3	Title_4	Emb_1	Emb_2	Emb_3
1	0.2750	0.014151	0	0	1	0.1	1	0	0	0	0	0	1
0	0.4750	0.139136	1	0	0	0.1	1	0	0	0	1	0	0
0	0.3250	0.015469	0	0	1	0.0	0	0	0	1	0	0	1
0	0.4375	0.103644	1	0	0	0.1	1	0	0	0	0	0	1
1	0.4375	0.015713	0	0	1	0.0	1	0	0	0	0	0	1

Image segmentation / background removal

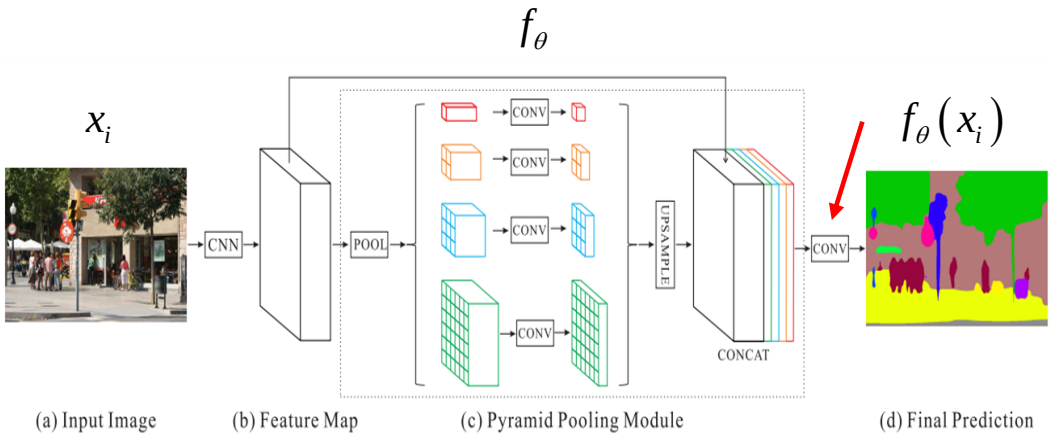
- Assume that for each image x_i, $i \in I$, from the dataset we have ground truth segmentation y_i.
- Each pixel $y_i(k_1, k_2)$ is 1 if the pixel is part of the required object and 0 if not.
- The output of the neural network is $f_\theta(x_i)$.
- The loss function optimizes the weights θ of the network by minimizing

$$-\frac{1}{\#\text{images}\#\text{pixels}}\sum_{i\in I}\sum_{k_1,k_2} y_i(k_1,k_2)\log h_\theta\left(f_\theta(x_i)(k_1,k_2)\right)+\left(1-y_i(k_1,k_2)\right)\log\left(1-h_\theta\left(f_\theta(x_i)(k_1,k_2)\right)\right)$$

Image segmentation / background removal

- Assume that for each image x_i, $i \in I$, from the dataset we have ground truth segmentation y_i.
- Each pixel $y_i(k_1, k_2)$ is 1 if the pixel is part of the required object and 0 if not.
- The output of the neural network is $f_\theta(x_i)$.
- The loss function optimizes the weights θ of the network by minimizing

$$-\frac{1}{\#\text{images}\#\text{pixels}}\sum_{i\in I}\sum_{k_1,k_2} y_i(k_1,k_2)\log h_\theta\left(f_\theta(x_i)(k_1,k_2)\right)+\left(1-y_i(k_1,k_2)\right)\log\left(1-h_\theta\left(f_\theta(x_i)(k_1,k_2)\right)\right)$$



Portrait segmentation - Examples

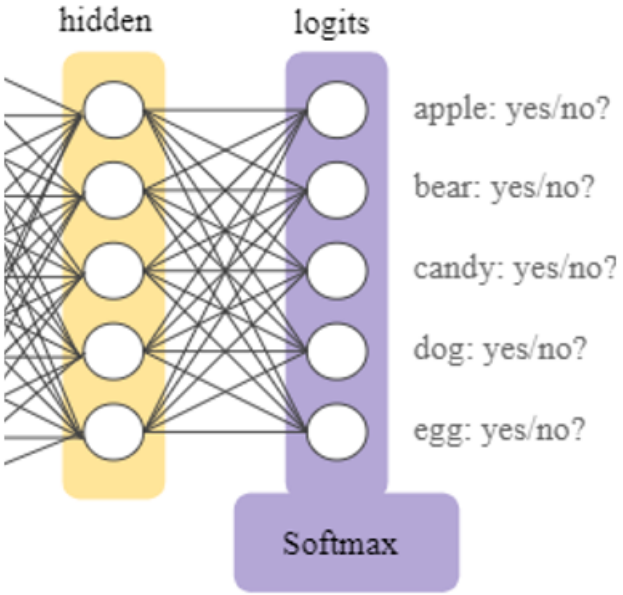
Portrait segmentation - Examples

Soft-Max: generalization of the logistic function

We have a classification problem with L classes, $L \geq 3$.

We use $\theta = \left\{ W \in M_{L \times n}, b \in \mathbb{R}^L \right\}$. We model, with W_k, the k-th row of W, $1 \leq k \leq L$,

$$\Pr\left(y = Y_k \mid x, \theta\right) := \frac{e^{-\left(\langle W_k, x \rangle + b_k\right)}}{\sum_{j=1}^{L} e^{-\left(\langle W_j, x \rangle + b_j\right)}}$$



Generalization of binary model. Why? Assume we have only two classes with $L = 2$.

$$\Pr\left(y = Y_1 \mid x, \theta\right) := \frac{e^{-\left(\langle W_1, x\rangle + b_1\right)}}{e^{-\left(\langle W_1, x\rangle + b_1\right)} + e^{-\left(\langle W_2, x\rangle + b_2\right)}} = \frac{1}{1 + e^{-\left(\langle W_2 - W_1, x\rangle + b_2 - b_1\right)}}$$

The associated loss function is the minimization of the negative of the log-likelihood

$$-\frac{1}{\#I}\sum_{i \in I} \log\left(\Pr\left(Y = y_i \mid \theta, x_i\right)\right).$$

Machine Learning – Some basic definitions

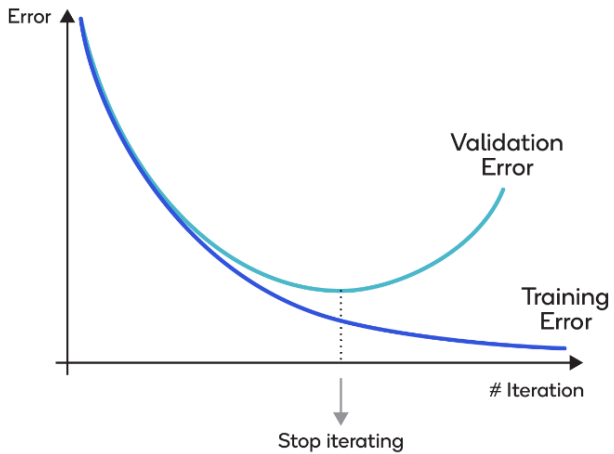
Basic Definitions

Labeled data – The quality of machine learning models highly(!) depends on the quality of the dataset. Does it cover the entire sampling space? Does it contain errors? Do we have sufficient samples.

Better data beats better models!

Training set – We train the model using a subset of the dataset which we call the training data.

Validation set - We have the option of using some 'holdout' data the model does not see. The validation data can be used for model hyper-parameter tuning. We will see many examples. One example is: stopping criteria for the optimization to avoid **over fitting** the training data and improving **generalization**.



Testing set – We use ‘ground truth’ testing data the model does not see to analyze its performance.

Typical example – random sampling of the dataset: 70% training, 10% validation, 20% testing.

Cross validation – A research technique where we randomly split the full dataset into (for example) 5 parts, build a model 5 times, each time testing on a different part after training on the remaining 4 parts. One can also randomly split the full set 5 times. We then compute the statistics of the models: average and variance of the error.

Inference –In applications, we apply the model to incoming unlabeled data.

K Fold CV, K=5

Confusion matrix

		Actual	
Predicted		**Categorical**	**Non-Categorical**
	Categorical	True Positive (TP)	False Positive (FP)
	Non-Categorical	False Negative (FN)	True Negative (TN)

FP = Type I Error, FN = Type II Error

We can obviously create a confusion matrix for arbitrary number of classes

		Actual				
Predicted		Class 1	Class 2	…	…	Class L
	Class 1					
	Class 2					
	Class L					

Optimally, outside the diagonal we hope to get small numbers/%.

Accuracy. The simplest form of measurement

$$\frac{TP+TN}{P+N}=\frac{TP+TN}{(TP+FN)+(TN+FP)}$$

Accuracy is problematic in "rare event cases". Suppose that we have a 'positive' outcome for 0.1% of the time. Then, a 'stupid' model that predicts 'negative' for each sample has accuracy 99.9%.

Sensitivity, Recall, True Positive Rate $\frac{TP}{P}=\frac{TP}{TP+FN}$

e.g.: what % of the 'positives' did the model find?

e.g., in document retrieval: relevant and retrieved / relevant

The 'stupid' model that always predicts 'negative' will have recall = 0.

Specificity, True Negative (False) Rate $\frac{TN}{N}=\frac{TN}{TN+FP}$

Averaged accuracy $\frac{1}{2}\left(\frac{TP}{P}+\frac{TN}{N}\right)$

Precision (Positive Prediction Value) $\frac{TP}{TP+FP}$

e.g. in document retrieval: Relevant and retrieved / all retrieved

False Positive Rate (fall out, false alarms) $\frac{FP}{N}=\frac{FP}{FP+TN}$

False Negative Rate (miss rate) $\frac{FN}{P}=\frac{FN}{TP+FN}$

F1 Score $2\frac{precision \times recall}{precision+recall}$