Example based explanation for interpretable machine learning

Contents

Introduction

Example-Based Explanations means that we use instances from the data to explain the model. It consists of 4 main method, *counterfactual explanations*, *adversarial examples*, *prototypes and criticisms* and *influential instances*. We will discuss each of them and provide code and examples.

Counterfactual explanations

Introduction

Counterfactual explanation is one of the example-based explanation for interpretable machine learning. The idea of counterfactual has been used in many fields like philosophy, psychology, and the social science (Verma et al. 2020). s. Philosophers like David Lewis published articles on the ideas of counterfactuals back in 1973. He explains that counterfactual can be write in a sentence with condition like: 'If it were the case that_, then it would be the case that…'. (Lewis 1973). Think an example, we are trying to test a loan approval algorithm, the accountable feature is income, age, credit score. If we have an instance with income 45,000, age 40, credit score 4, the prediction is No, we cannot get a loan, and there is another instance with income 55,000, age 40, credit score 4, the prediction is yes, and we call the second instance is counterfactual instance of the first one. The counterfactual method could provide useful information for loaner to get their loan approved.

Explanation

One thing about the counterfactual explanation is that we want to achieve different outcome while making the minimal change to the input data. A formular suggest by Wachter et.all (2017) as cited in Molnar (2020):

$$L(x, x', y', \lambda) = \lambda \cdot (\hat{f}(x') - y')^2 + d(x, x')$$

x is the origin input data, X' is modified input, y' desired outcome, λ is a parameter balance the distance in prediction (first term) against the distance in the feature values(second term), we need to minimize the loss between desired outcome and predicted outcome(by x') while keep x' close to x.

when λ is small, minimize the distance between input features become important, like the loan approval example, only change one feature income we can get a loan approval. If λ is big, minimize the distance between the original prediction and desired outcome become importance, so we want to get loan approval without worry about how every input feature changes too much.

Instead of control the λ value, the author of the method suggests a threshold ϵ:

$$|\hat{f}(x') - y'| \leq \epsilon$$

This means predicted outcome by the modified input (counterfactual) must close to the desired outcome, within a user defined threshold.

To implement the formular into an algorithm, we have

$$d(x, x') = \sum_{j=1}^{p} \frac{|x_j - x'_j|}{MAD_j}$$

distance is sum of all P feature-wise distance. MAD is equivalent of the variance of a feature, but more robust to outliers.

And

$$\arg\min_{x'} \max_{\lambda} L(x, x', y', \lambda)$$

This means we have preset ϵ, we need x' that minimize the loss, increase λ until we find optimum.

Advantages:

- Counterfactual method is good, because it can help us make decision without even looking at the original data or model, all we need is the model's prediction function. This is useful when someone offer explanation for users who must protect their data.
- The method works with systems that do not use machine learning(prediction based on handwritten rules), and it is relatively easy to implement (Molnar 2020).

Disadvantages:

- There could be multiple counterfactual explanations for one instance, which one to choose. So in reality, we often have to use our domine knowledge to decide the optimal option, which is adding more work to the method.
- No guarantee a solution is found under a given tolerance ϵ.
- Lack of software implementation.

Boston dataset example

Our goal is to change the most important feature's value to make prediction different. Since Boston dataset's prediction is continuous value, we are going to use the base value (average of predicted value) as the boundary, this means if the predicted value greater than base value, we want to change it to lower, vice vera.

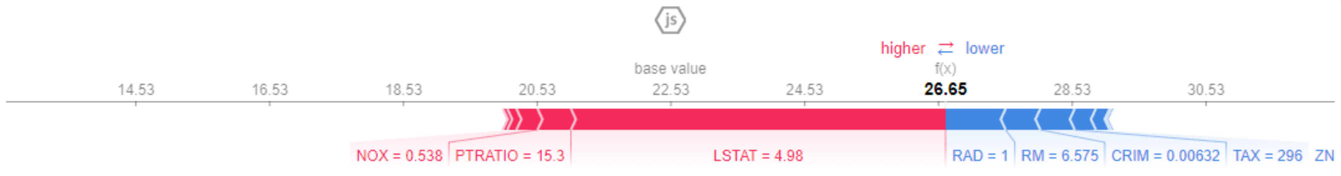
For the first instance:

```
[123] import numpy as np
      input = X.loc[[0],]
```

```
[124] input
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.09	1.0	296.0	15.3	396.9	4.98

```
[122] shap.initjs()
      shap.plots.force(shap_values[0])
```



The most important feature is 'LSTAT', the base value is 22.53, the prediction is 26.65. We want to change the value of LSTAT so that the prediction will be lower than 22.53.

```
input['LSTAT'] = input['LSTAT'] * 3
```

```
model.predict(input)
```

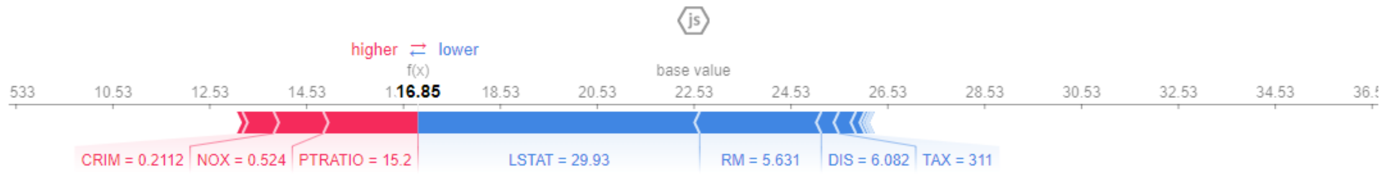
```
array([20.904549], dtype=float32)
```

We trippled the LSTAT 's value and **prediction successfully dropped to 20.90 (lower than base value).**

Let's try another example:

For instance 8,

```
shap.initjs()
shap.plots.force(shap_values[8])
```



```
input = X.loc[[8],]
```

```
input
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93

```
model.predict(input)
```

```
array([16.845892], dtype=float32)
```

```
input['LSTAT'] = input['LSTAT'] / 4
```

```
model.predict(input)
```

```
array([22.74664], dtype=float32)
```

The predicted value was successfully increased from 16.85 to 22.74 (greater than base value).

Adversarial Examples

Concept

Adversarial examples are a bit like counterfactual instances, it is an instance with small change led to wrong prediction.

Szegedy et. al(2013) used a gradient based optimization to find adversarial examples for deep neural network. Example show in figure 1:

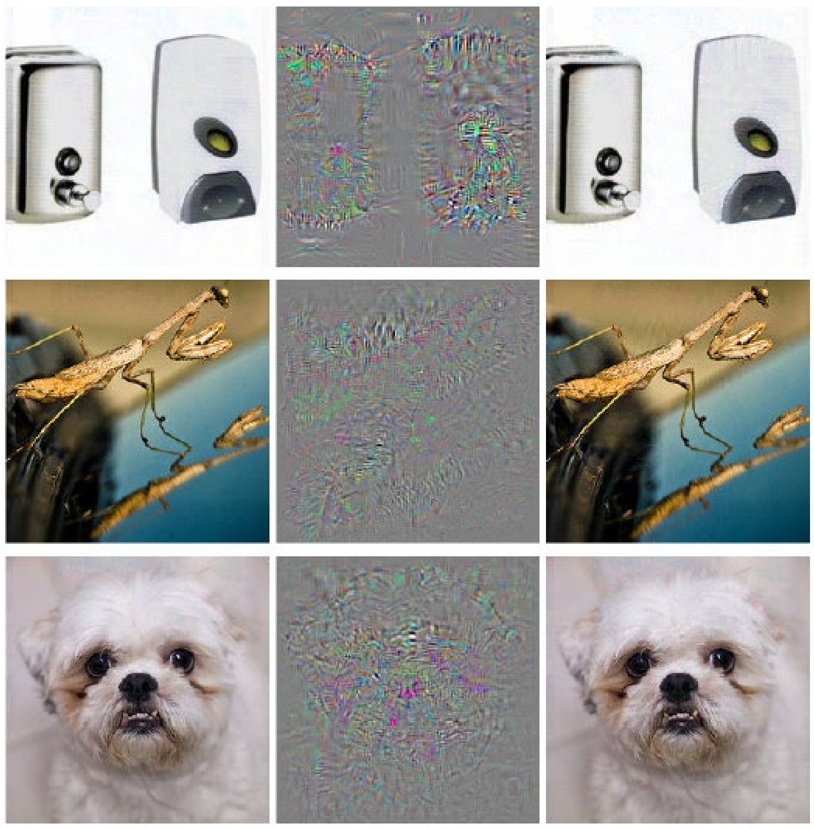


Figure 1 Adversarial examples for AlexNet by Szegedy et. al (2013). First column on the left are the original pictures which are correctly predicted, the middle column is the error added to the images, then the last column shows the picture after modification which are predicted as 'Ostrich'.

The formular used for generating adversarial examples which is identical to the counterfactual explanation formular:

$$loss(\hat{f}(x+r), l) + c \cdot |r|$$

x is the original input, r is error term, l is desired outcome, c is used to balance the distance between inputs(r) and distance between predictions.

Goodfellow, Shlens and Szegedy (2014) introduced a **Fast Gradient Sign Method** (FGSM) as follow:

$$x' = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y))$$

∇xJ is the gradient of loss function with respect to the original data x, y is the prediction, θ is the model parameter.

The important to know about FGSM is to add the noise whose direction is the same as the gradient of the cost function with respect to the **data**, the noise scaled with ϵ . This is different to gradient decent method; the gradient of the cost function is respect to the **weight**.

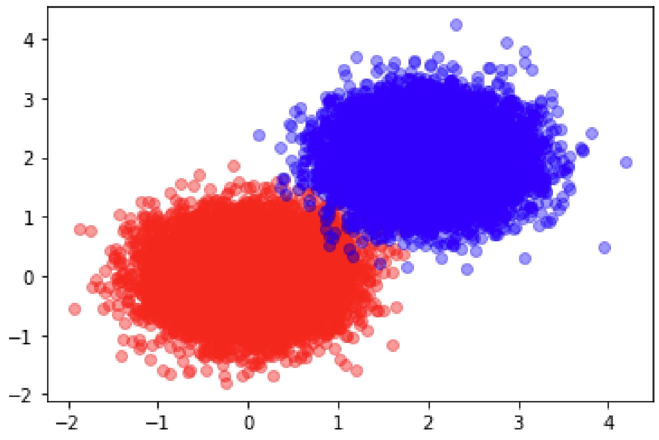
First example

Tsui (2018) showed how adversarial example can alter the prediction with simple code example (full code see Appendix A):

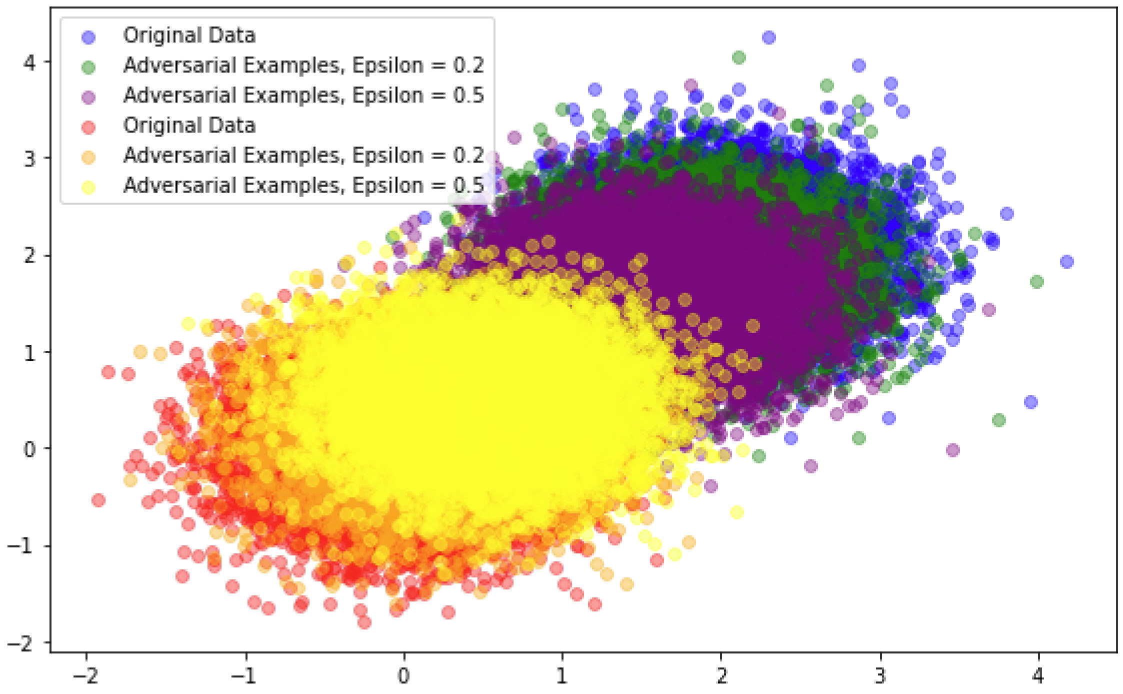
First, we build 2 types of points in a 2-D dimension, **red** dots are class 1, **blue** dots are class 0, then we define a **logistic**_regression object as original model. For the

```
plt.scatter(X[10000:, 0], X[10000:, 1], color='red', alpha=0.4, label='Original Data')
plt.scatter(X[:9999, 0], X[:9999, 1], color='blue', alpha=0.4, label='Original Data')
```

```
<matplotlib.collections.PathCollection at 0x7f6431882460>
```



By passing different epsilon we get different adversarial examples:



```
Error Rate without adversarial examples: 0.03145
Error Rate with adversarial examples, epsilon = 0.2: 0.0798
Error Rate with adversarial examples, epsilon = 0.5: 0.222
```

We can see how adversarial examples moved compare to the original ones. The error rates getting higher when epsilon increase. 22% error rates when epsilon is 0.5.

Second example

The following image adversarial example is from TensorFlow tutorial

We use MobileNetV2 model which is pretrained on ImageNet.

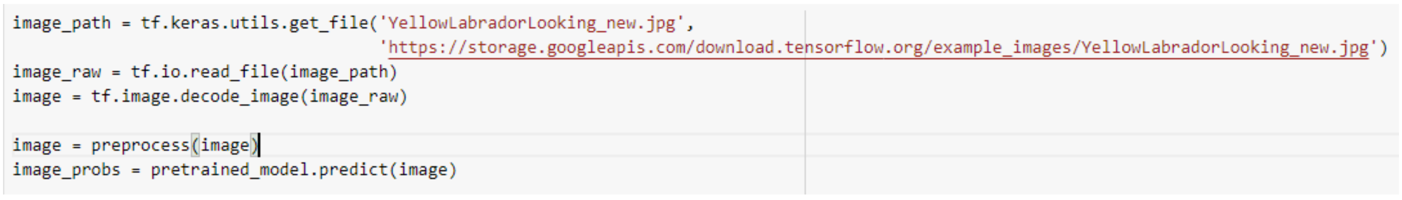
```
import tensorflow as tf
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams['figure.figsize'] = (8, 8)
mpl.rcParams['axes.grid'] = False
```

```
# the model is MobileNetV2 model, pretrained on ImageNet.
pretrained_model = tf.keras.applications.MobileNetV2(include_top=True,
                                                     weights='imagenet')
pretrained_model.trainable = False

# ImageNet labels
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions
```

Then we create helper function to preprocess the image so that it can be inputted in MobileNetV2.

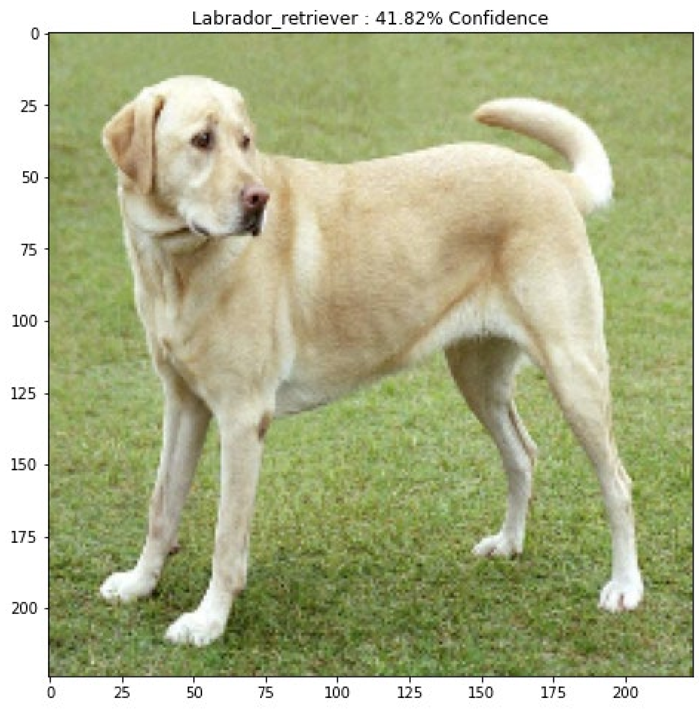
```
def preprocess(image):
  image = tf.cast(image, tf.float32)
  image = tf.image.resize(image, (224, 224))
  image = tf.keras.applications.mobilenet_v2.preprocess_input(image)
  image = image[None, ...]
  return image

# Helper function to extract labels from probability vector
def get_imagenet_label(probs):
  return decode_predictions(probs, top=1)[0][0]
```

The sample image we use is Labrador Retriever by Mirko CC-BY-SA 3.0 from Wikimedia Common.



```
image_path = tf.keras.utils.get_file('YellowLabradorLooking_new.jpg',
                                     'https://storage.googleapis.com/download.tensorflow.org/example_images/YellowLabradorLooking_new.jpg')
image_raw = tf.io.read_file(image_path)
image = tf.image.decode_image(image_raw)

image = preprocess(image)
image_probs = pretrained_model.predict(image)
```

```
plt.figure()
plt.imshow(image[0] * 0.5 + 0.5)  # To change [-1, 1] to [0,1]
_, image_class, class_confidence = get_imagenet_label(image_probs)
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```



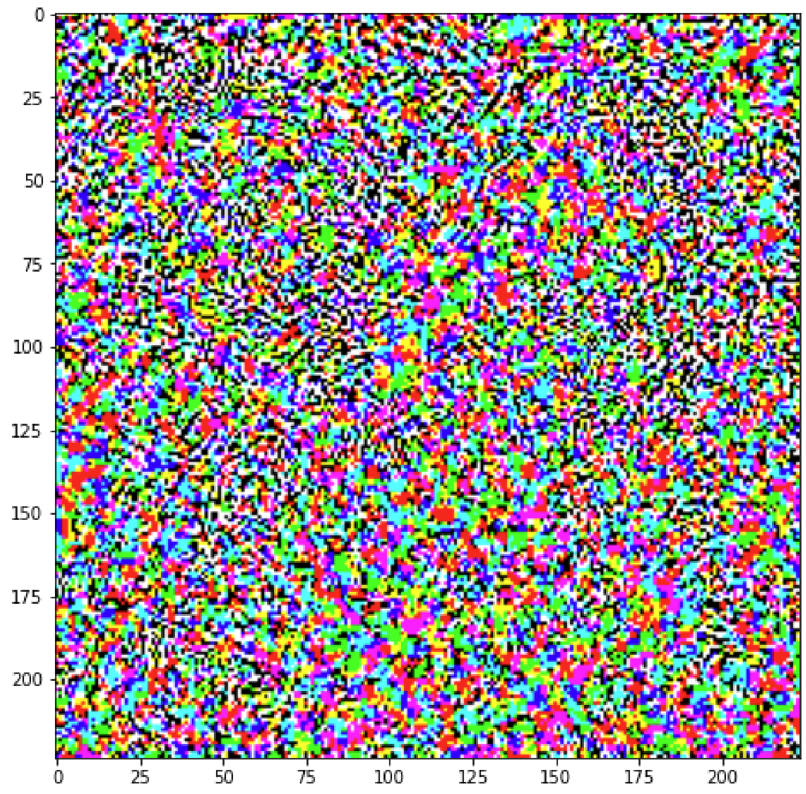
Then we create adversarial image,

Implementing fast gradient sign method, the gradients are taken with respect to the image.

```
loss_object = tf.keras.losses.CategoricalCrossentropy()

def create_adversarial_pattern(input_image, input_label):
  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = pretrained_model(input_image)
    loss = loss_object(input_label, prediction)

  # Get the gradients of the loss w.r.t to the input image.
  gradient = tape.gradient(loss, input_image)
  # Get the sign of the gradients to create the perturbation
  signed_grad = tf.sign(gradient)
  return signed_grad
```

The perturbations as image:

```
labrador_retriever_index = 208
label = tf.one_hot(labrador_retriever_index, image_probs.shape[-1])
label = tf.reshape(label, (1, image_probs.shape[-1]))

perturbations = create_adversarial_pattern(image, label)
plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to [0,1]
```

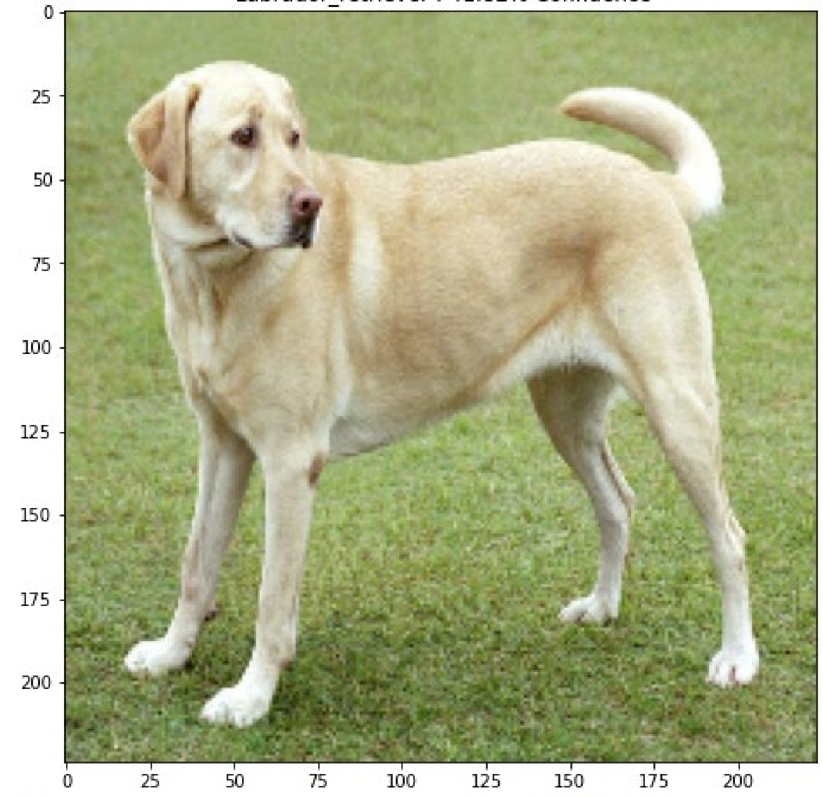


Now we can plot the images with different epsilons:

```
def display_images(image, description):
  _, label, confidence = get_imagenet_label(pretrained_model.predict(image))
  plt.figure()
  plt.imshow(image[0]*0.5+0.5)
  plt.title('{} \n {} : {:.2f}% Confidence'.format(description,
                                                   label, confidence*100))
  plt.show()
```

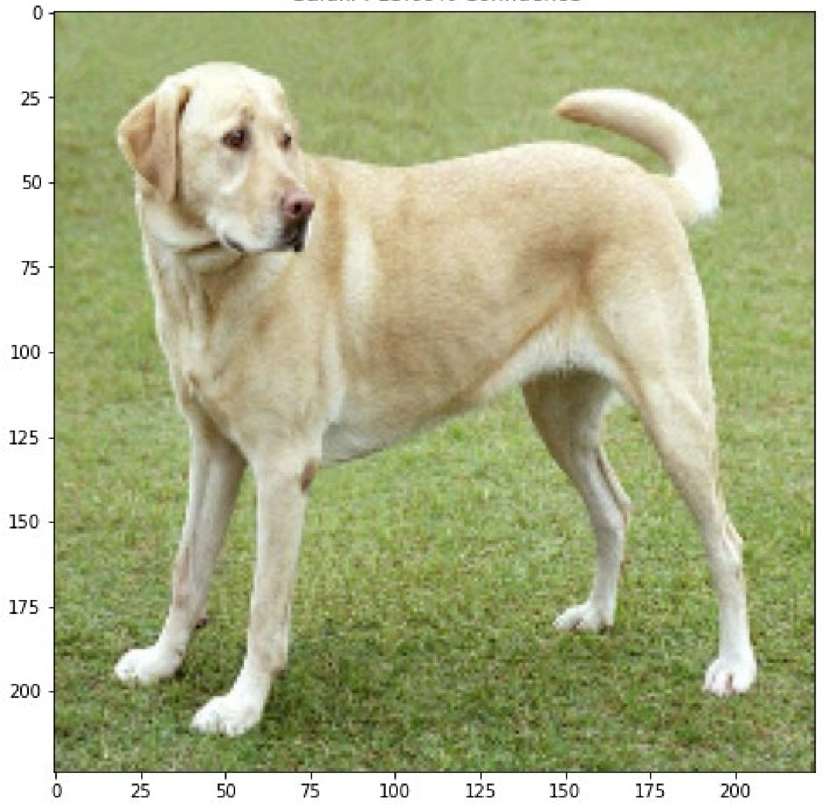
```
epsilons = [0, 0.01, 0.1, 0.15]
descriptions = [('Epsilon = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]

for i, eps in enumerate(epsilons):
  adv_x = image + eps*perturbations     # perturbation = signed_gradient
  adv_x = tf.clip_by_value(adv_x, -1, 1)   # Clips tensor values to a specified min and max.(-1,1)
  display_images(adv_x, descriptions[i])
```

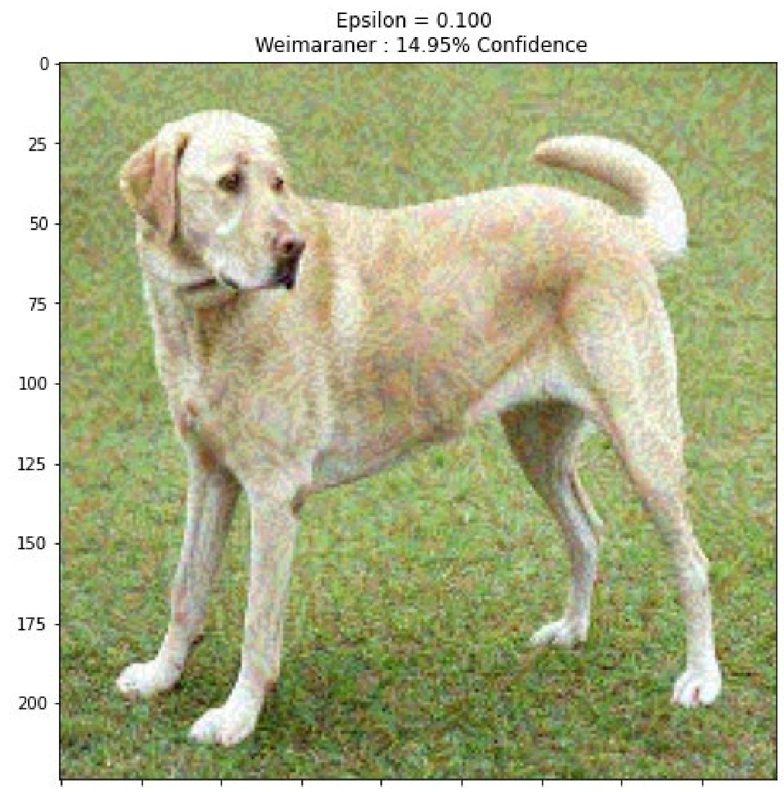
Input
Labrador_retriever : 41.82% Confidence

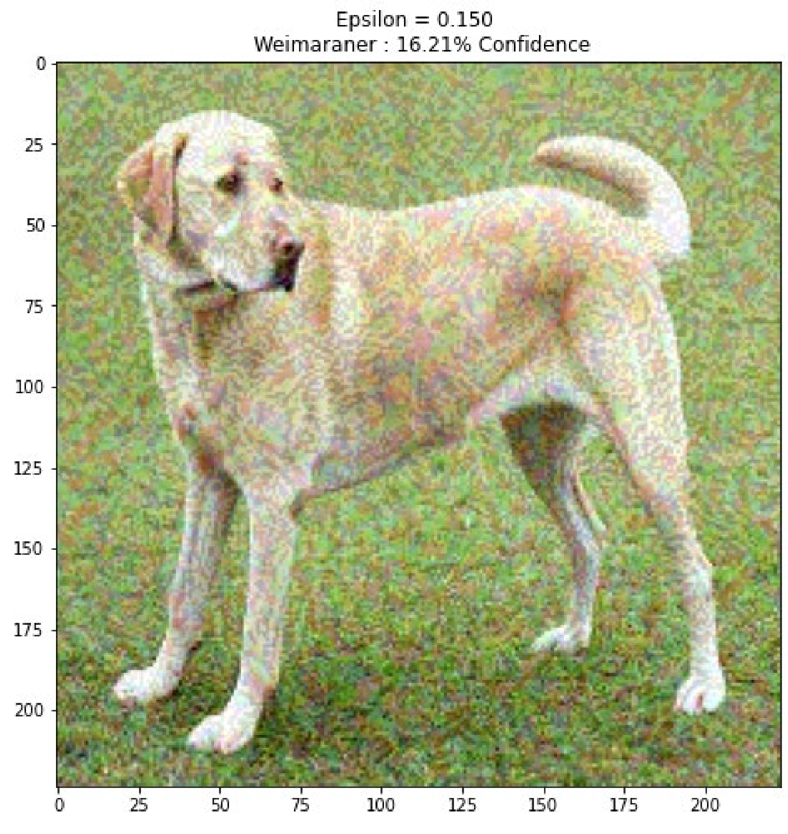


```
1/1 [==============================] - 0s 84ms/step
```

Epsilon = 0.010
Saluki : 13.09% Confidence







Like example 1, as the value of epsilon increase, error rates increase, the model is more likely to make mistake, it has more confidence to predict the picture as Weimaraner as opposite to Labrador. However, more perturbation makes the picture easier to identify as an adversarial attack by human eyes.

Boston data example

Like example one, for Boston dataset, we can also modify the input data, so the perdition error increases.

We train a linear regression model since it's easier to get the weights.

```
[37] model = LinearRegression()
```

```
[38] model.fit(train, labels_train)

     LinearRegression()
```

```
[40] MSE = np.mean((model.predict(test) - labels_test) ** 2)
     print('MSError', MSE)

     MSError 0.11119851023826761
```

The MSE (mean square error) is 0.11.

Then we find adversarial examples to modify the input data.

```
def adversial_samples(X, Y, model, epsilon=0.00001):
    dlt = model.predict(X).T - Y      # delta
    # the sign function returns -1 if x < 0, 0 if x==0, 1 if x > 0.
    # delta is the gradient of cost with respect to data X ,hence, (Y_Prediction – Y_True)W
    direction = np.sign(np.matmul(dlt.reshape(dlt.shape[0],1),
                                  model.coef_.reshape(1,model.coef_.shape[0])))

    return X + epsilon * direction, Y
```

```
Xadv, Y = adversial_samples(test, labels_test, model, epsilon=0.2)
print('MSE with adversarial examples, epsilon = 0.2:', np.mean((model.predict(Xadv) - labels_test) ** 2))
Xadv2, Y = adversial_samples(test, labels_test, model, epsilon=0.5)
print('MSE with adversarial examples, epsilon = 0.5:', np.mean((model.predict(Xadv2) - labels_test) ** 2))
```

```
MSE with adversarial examples, epsilon = 0.2: 0.27756958970312
MSE with adversarial examples, epsilon = 0.5: 0.7004019075490251
```

When epsilon is 0.2, the MSE increases to 0.28, and when epsilon is 0.5, the MSE increase to 0.7.

How to defend an adversarial attack

Now we come up the question, if the machine leaning model is so easy to get destroyed by adversarial attack, how we are going to prevent that happen? Molnar (2020) suggest that we need to be proactive about the attack.

We can train model with adversarial examples, this makes the model more robust to attacks, this especially useful in white box attacks (attackers know the model well, they can easily find adversarial examples through gradient of the network loss function). However, this is not very effective when met black box attacks (attackers are the ones did not know the underline model, they may just use input and output to generate their own weights and loss function to approximate adversarial examples), since the model used to find adversarial examples are different. We also need to understand our model, which features are more vulnerable to specific class label's attack, this can be learned through various interpretable machine leaning method. Then we can constantly test and retrain, to make the model more robust to attacks.

Prototypes and Criticisms

Concept

A **prototype** is the data points that represent the whole dataset (think it like the k-mean clustering), while **criticisms** are the data points which are not well represented by the prototypes. There are many ways to find prototypes in the data. Kim, Khanna, and Koyejo (2016), shows a method called MMD-critic which combines prototype and criticism together. It uses Kernel function to calculate the data densities. If the data closer to each other, the function return a value closer to 1, otherwise close to 0.

$$k(x, x') = exp\left(\gamma||x - x'||^2\right)$$

x is the chosen data points, x' is every other data points. The Euclidean distance is used in this function, γ is a scaling parameter.

Then it measures the **maximum mean discrepancy** (MMD):

$$MMD^2 = \frac{1}{m^2}\sum_{i,j=1}^{m} k(z_i, z_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(z_i, x_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(x_i, x_j)$$

m is the number of points chosen as prototypes, n is the number of other points., z is the prototypes, x is the other points.

The closer the MMD^2 to zero, it means the prototypes are better fits the data.

We can plot the MMD^2 use R (full code see Appendix B):

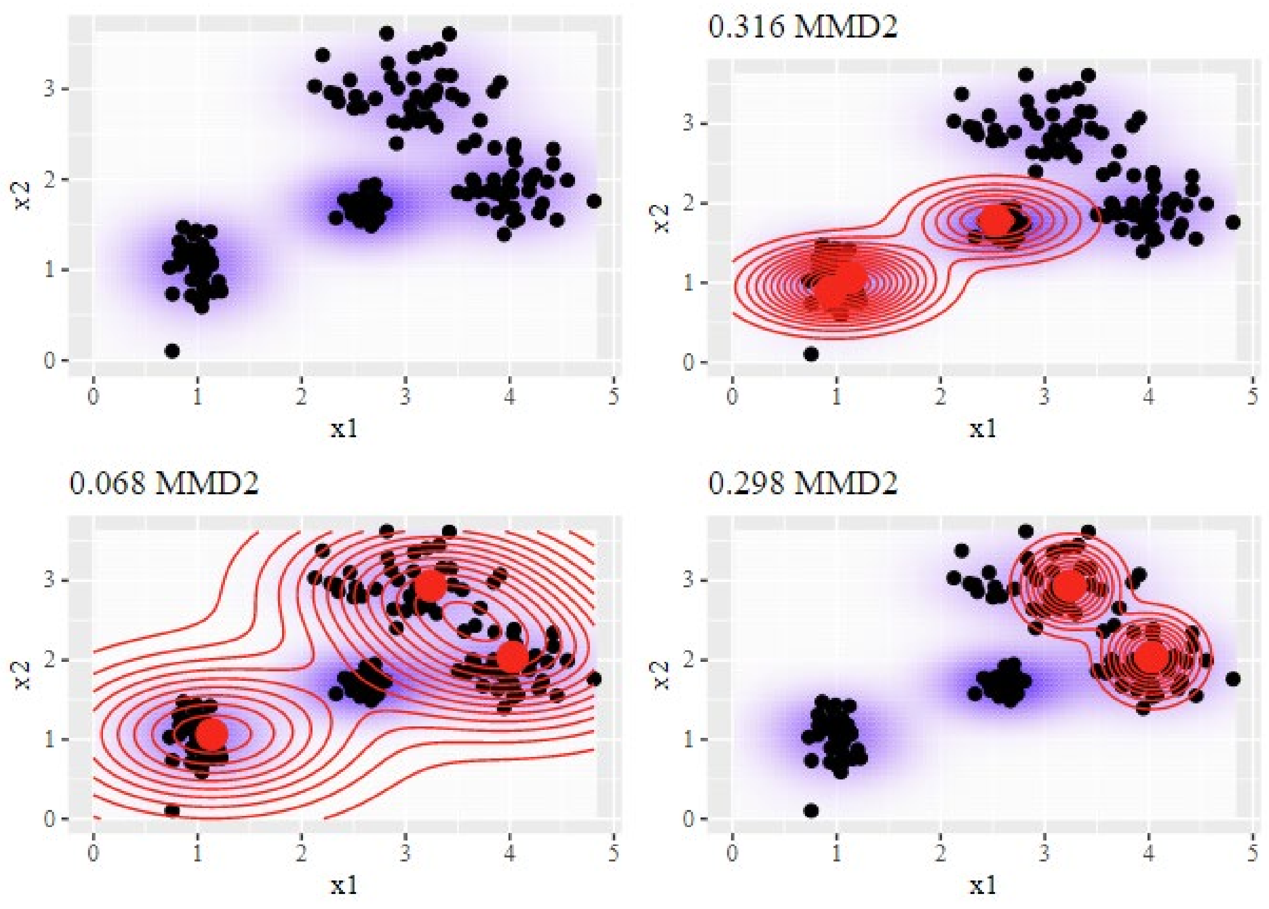


Figure 2 different MMD² shows how prototype represent the data, the closer to 0, the better fit. (Image from Molnar (2020)

We can see that the graph in bottom left have the MMD² close to 0 and it is the best fit in all 3.

For Criticisms, it uses witness function which shows how much two density estimates differ at a chosen point.

$$witness(x) = \frac{1}{n}\sum_{i=1}^{n} k(x, x_i) - \frac{1}{m}\sum_{j=1}^{m} k(x, z_j)$$

If witness function gives a **positive** number, it might indicate a situation which many data points around x but no prototypes nearby(underestimates), if witness function gives a **negative** number there is a prototype with not many data around it(overestimates).

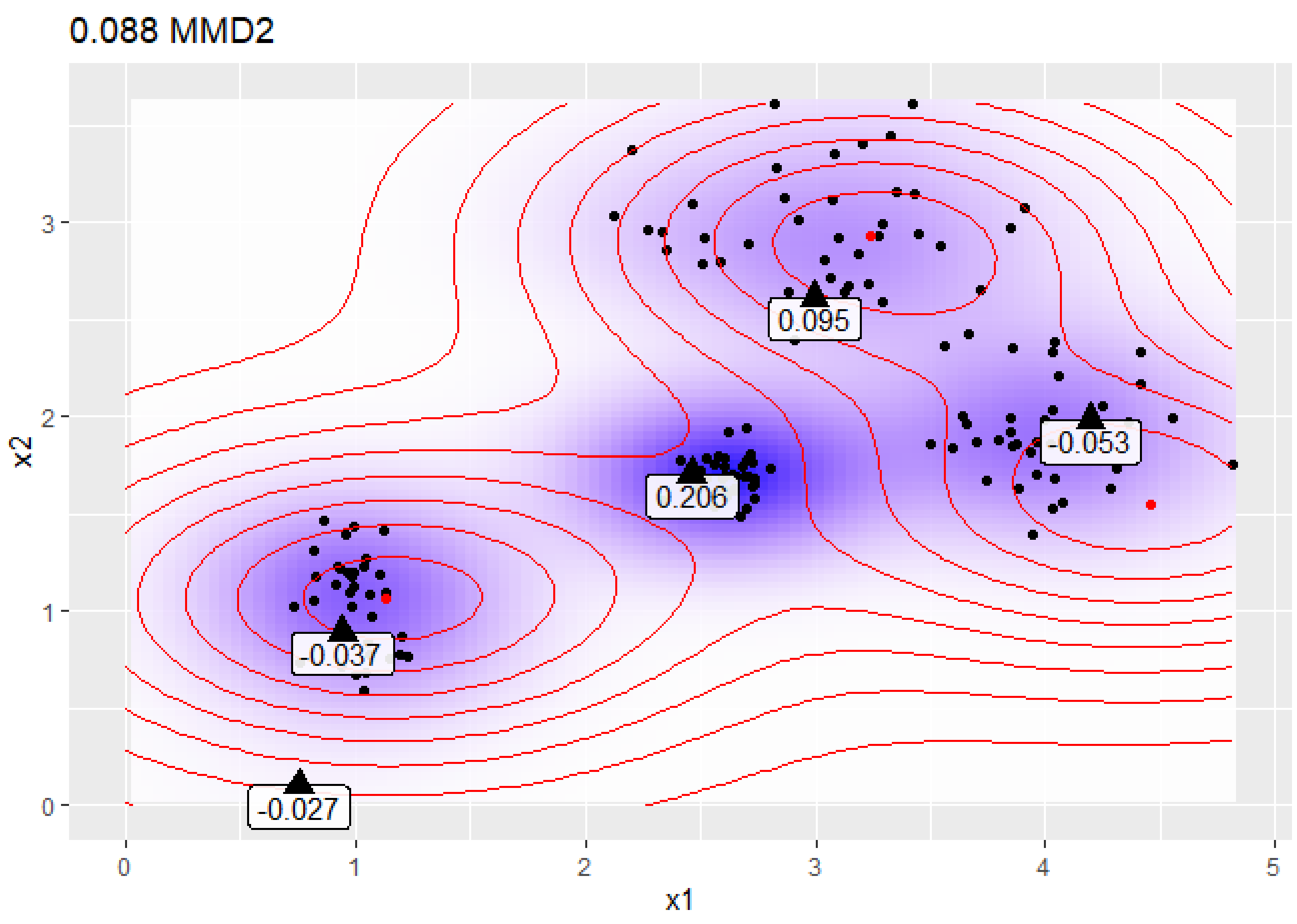


Figure 3 witness function values for different chosen datapoints

In Figure 3, the point in the middle which having the positive witness function value (0.206) indicates there are many data points nearby but no prototype to represent them. The point on the far right with the smallest negative number, shows a prototype near (red dot) by but not in the biggest crowed area.

The extreme witness function value (negative/positive) indicates the criticisms.

The witness function also helps us to identify prototypes whether they are well represented the data.

Boston data example

We will demonstrate phototypes and test criticisms for Boston dataset using R.

We use kmeans to find prototypes(centers).

```
df <- read.csv('Boston.csv',row.names = 1)
df <- scale(df)
model <- kmeans(df, centers = 3, nstart = 25)
centers <- model$centers
```

Then define the *mmd2* and *witness* functions:

```
radial = function(x1, x2, sigma = 1) {
  dist = sum((x1 - x2)^2)        # why not use euclidean <- function(a,b) sqrt(sum((a-b)^2))
  exp(-dist/(2*sigma^2))
}

cross.kernel = function(d1, d2) {
  kk = c()
  for (i in 1:nrow(d1)) {
    for (j in 1:nrow(d2)) {
      res = radial(d1[i,], d2[j,])
      kk = c(kk, res)
    }
  }
  mean(kk)
}

mmd2 = function(d1, d2) {
  cross.kernel(d1, d1) - 2 * cross.kernel(d1, d2) + cross.kernel(d2,d2)
}

# mmd2(df,centers)

witness = function(x, dist1, dist2, sigma = 1) {
  k1 = apply(dist1, 1, function(z) radial(x, z, sigma = sigma))
  k2 = apply(dist2, 1, function(z) radial(x, z, sigma = sigma))
  mean(k1) - mean(k2)
}
```

We select random points to test for criticisms:

```
w.points.indices = c(15, 2, 60, 19, 10)
wit.points = df[w.points.indices,]

# use witness function
for (i in 1:nrow(wit.points)){
  print(i)
  print(witness(wit.points[i,], df, centers, sigma = 1))
}
```

And we got witness value for the 5 selected points:

```
1
-0.03594697
2
-0.1192489
3
0.01027594
4
-0.002090971
5
-0.007001356
```

It seems like no extreme positive/negative values here, so they are not criticisms.

As demonstrated, we can interpret individual instance whether it is represented by phototypes or not.

Advantages and disadvantages

MMD-critic can help us understand our model, build interpretable model especially useful when encounter black box models. It works with any type of data or machine leaning models.

However, we must choose the number of Prototypes and criticisms ourselves, how many prototypes is better? We can plot the number of prototypes(x-axis) respect to the MMD^2(y-axis) to see when the curve is flattened. And if we have many irrelevant features, it will affect our prototype and criticisms, we may have to select reduce feature dimensions first.

Influential instance

Concept

Now we are changing our focus for a bit, instead of focus on important features, we move on to a single instance of a dataset. If an instance affects our prediction greatly, we call it influential instance. Think about outliers, if the outlier affecting the model's parameters and prediction by a great margin, we can say this outlier is an influential instance (figure 4). By analyzing the influential instance, we can find the reason why our model affected by it. This can not only help us interpret our model better and but also make the model more robust.

We can show a example of linear model in R:

```r
# graph for outliers affecting linear model
library(ggplot2)
library(dplyr)
set.seed(42)
n = 50
x = rnorm(mean = 1, n = n)
x = c(x, 8)
y = rnorm(mean = x, n = n)
y = c(y, 7.2)
df = data.frame(x, y)
ggplot(df) + geom_histogram(aes(x = x)) +
  scale_x_continuous("Feature x") +
  scale_y_continuous("count")

df2 = df[-nrow(df),]
df3 = rbind(df2, data.frame(x = 8, y = 0))

df3$regression_model = "with influential instance"
df2$regression_model = "without influential instance"
df.all = rbind(df2, df3)


text.dat = data.frame(x = c(8), y = c(0), lab = c("Influential instance"),
                      regression_model = "with influential instance")

ggplot(df.all, mapping = aes(x = x, y = y, group = regression_model, linetype = regression_model)) +
  geom_point(size = 2)+
  geom_smooth(method='lm',formula=y~x, se = FALSE, aes(color = regression_model), fullrange = TRUE) +
  geom_label(data = text.dat, aes(label = lab), hjust = 1, nudge_x = -0.2, vjust = 0.3) +
  scale_x_continuous("Feature x") +
  scale_y_continuous("Target y") +
  scale_color_discrete("Model training") +
  scale_linetype_discrete("Model training")
```

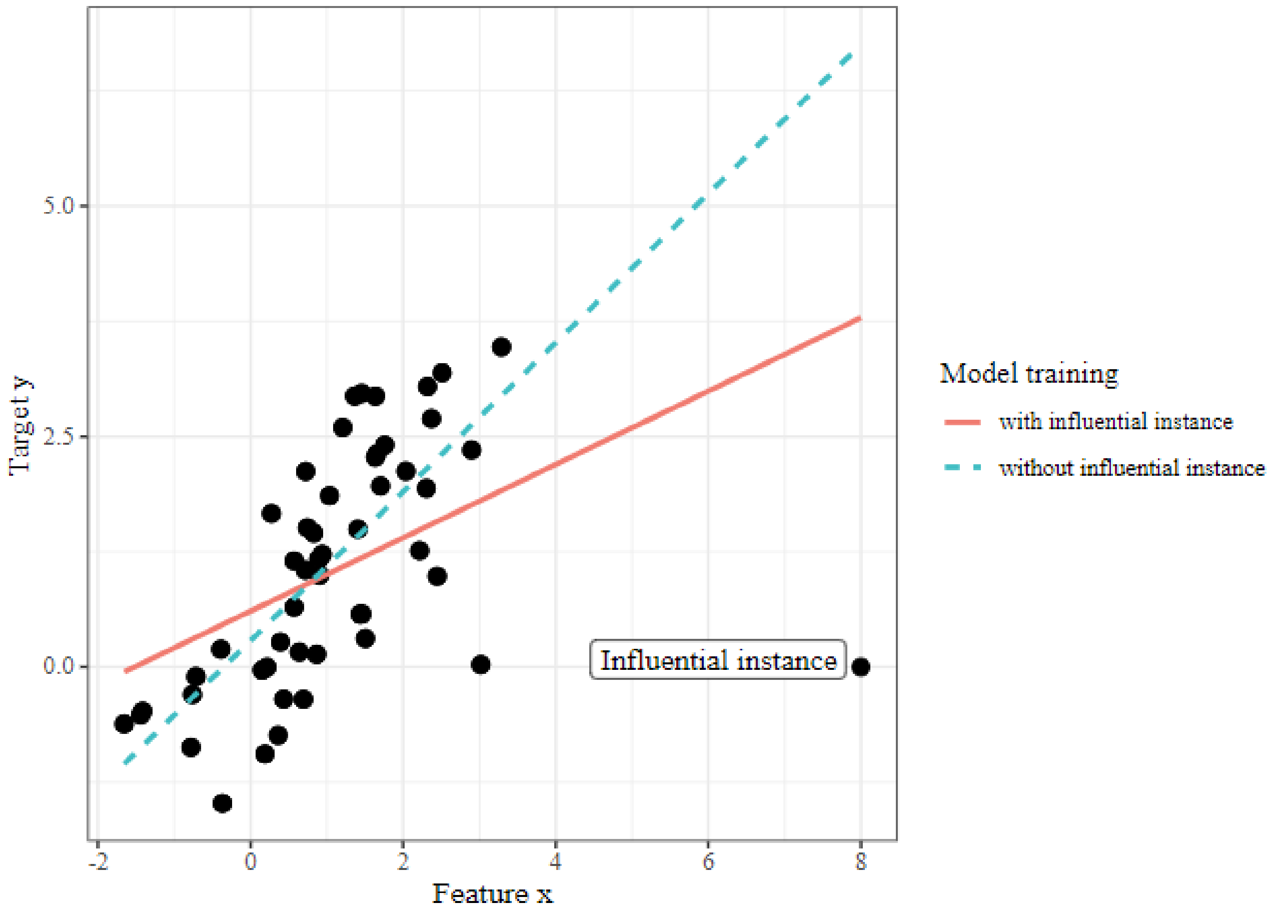


Figure 4 Outliers that changed the fitting line in the model. (Example from 'Interpretable Machine Learning A Guide for Making Black Box Models Explainable' Molnar, C 2020)

There are two ways to obtain the influential instance, one is Deletion Diagnostics, comparing the performance difference between deleting or not deleting an instance. The other one is a method involving gradients of the model parameters and upweight a data instance (Molnar, C 2020). We start with the first one

Deletion Diagnostics method

First, there are two well-known measures need to mention here, one is **DFBETA**, which measures how an influential instance affect the parameters. The formular is:

$$DFBETA_i = \beta - \beta^{(-i)}$$

β is the weights from the trained model, $\beta^{(-i)}$ means weight without influential instance. DFBETA works for model like Logistic regression and Neural networks, since they use trained weights.

The other measure is Cook's distance (Cook 1977 cited in Molnar, C 2020), it is defined similar as sum of squared error, the predicted error will be between original prediction and the perdition without influential instance:

$$D_i = \frac{\sum_{j=1}^{n}(\hat{y}_j - \hat{y}_j^{(-i)})^2}{p \cdot MSE}$$

Here p is the number of features. j is instances. The formular unfortunately only can be used with linear models(because MSE), if we want to use for any models, the simplest influence measure will be written as follow (Molnar, C 2020):

$$\text{Influence}^{(-i)} = \frac{1}{n}\sum_{j=1}^{n}\left|\hat{y}_j - \hat{y}_j^{(-i)}\right|$$

For the impact on a single instance(j) it will be:

$$\text{Influence}_j^{(-i)} = \left|\hat{y}_j - \hat{y}_j^{(-i)}\right|$$

For the previous linear model:

We can calculate:

```
# model df3 is original data, df 2 is influential moved data
df2.lm <- lm(y~x,df2)
df3.lm <- lm(y~x,df3)
new <- data.frame(x=df3$x)

# influential measure of instance (8,0)
mean(abs(predict(df3.lm,new)-predict(df2.lm,new)))

 [1] 0.4222079

# average of original prediction
mean(predict(df3.lm,new))

[1] 1.044147

> 0.42/1.04
[1] 0.4038462
```

Ideally, we need to use test data for prediction, but in this simple example, we just use df3(original data for predict). The **influence measure** of that influence instance (8,0) is 0.42. This means on average, the prediction will change by 0.42. Considering the average of original prediction is 1.04, the change caused by the influential instance are 40%! This is just a fake dataset, but we can still see the power of just one instance.

We also can see all other instance's influence measure:

```
original_predict <- predict(df3.lm,new)
influential <- c()
for (i in 1:length(df3$x)){
  dfi <- df3[-i,]
  dfi.lm <- lm(y~x,dfi)
  influential <- c(influential,mean(abs(original_predict -predict(dfi.lm,new))))
  }
print(influential)
```

```
> print(influential)
 [1] 0.0247026460 0.0235854364 0.0358409180 0.0204629976 0.0066005804 0.0041852254 0.0354353114 0.0005875343
 [9] 0.0456123266 0.0048742814 0.0088243179 0.0428513705 0.0062098260 0.0247744919 0.0162289299 0.0337276094
[17] 0.0032534957 0.0194335367 0.0178528441 0.0322556079 0.0249205743 0.0309723411 0.0103374890 0.0047227005
[25] 0.0147395307 0.0065452392 0.0122646678 0.0158859962 0.0122572434 0.0314475105 0.0357105810 0.0136230136
[33] 0.0144335363 0.0103321273 0.0178999902 0.0110895893 0.0149160962 0.0151677915 0.0168031823 0.0168019828
[41] 0.0302571684 0.0142072497 0.0221550354 0.0202949732 0.0464358619 0.0120838940 0.0350729512 0.0130343247
[49] 0.0037697926 0.0209494140 0.4222078658
```

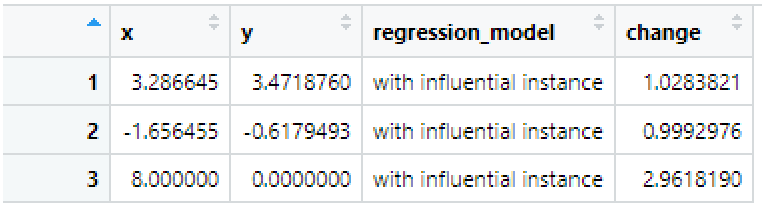
And the mean influence measure of all possible deletions:

```
> mean(influential)
[1] 0.02703214
```

This gives us the idea of how each point influence the model. We can also use this information to detect the problems in our dataset.

For interpreting purpose, we can **Use influential instance to explain individual instance.** First**,** we got our influential instance (8,0), we then find the top 3 data instance that affected by the influential instance the most (Figure 5).

```
# which instance affected by influential instance the most
df3$change <- abs(predict(df3.lm,new)-predict(df2.lm,new))
top3 <- top_n(df3,3)
```



	x	y	regression_model	change
1	3.286645	3.4718760	with influential instance	1.0283821
2	-1.656455	-0.6179493	with influential instance	0.9992976
3	8.000000	0.0000000	with influential instance	2.9618190

Figure 5 the top 3 instance influenced (by 8,0) the most.

```
ggplot(df.all, mapping = aes(x = x, y = y, group = regression_model, linetype = regression_model)) +
  geom_point(size = 2)+
  geom_smooth(method='lm',formula=y~x, se = FALSE, aes(color = regression_model), fullrange = TRUE) +
  geom_label(data = text.dat, aes(label = lab), hjust = 1, nudge_x = -0.2, vjust = 0.3) +
  scale_x_continuous("Feature x") +
  scale_y_continuous("Target y") +
  scale_color_discrete("Model training") +
  scale_linetype_discrete("Model training") +
  geom_vline(xintercept=top3$x, colour = 'orange')
```

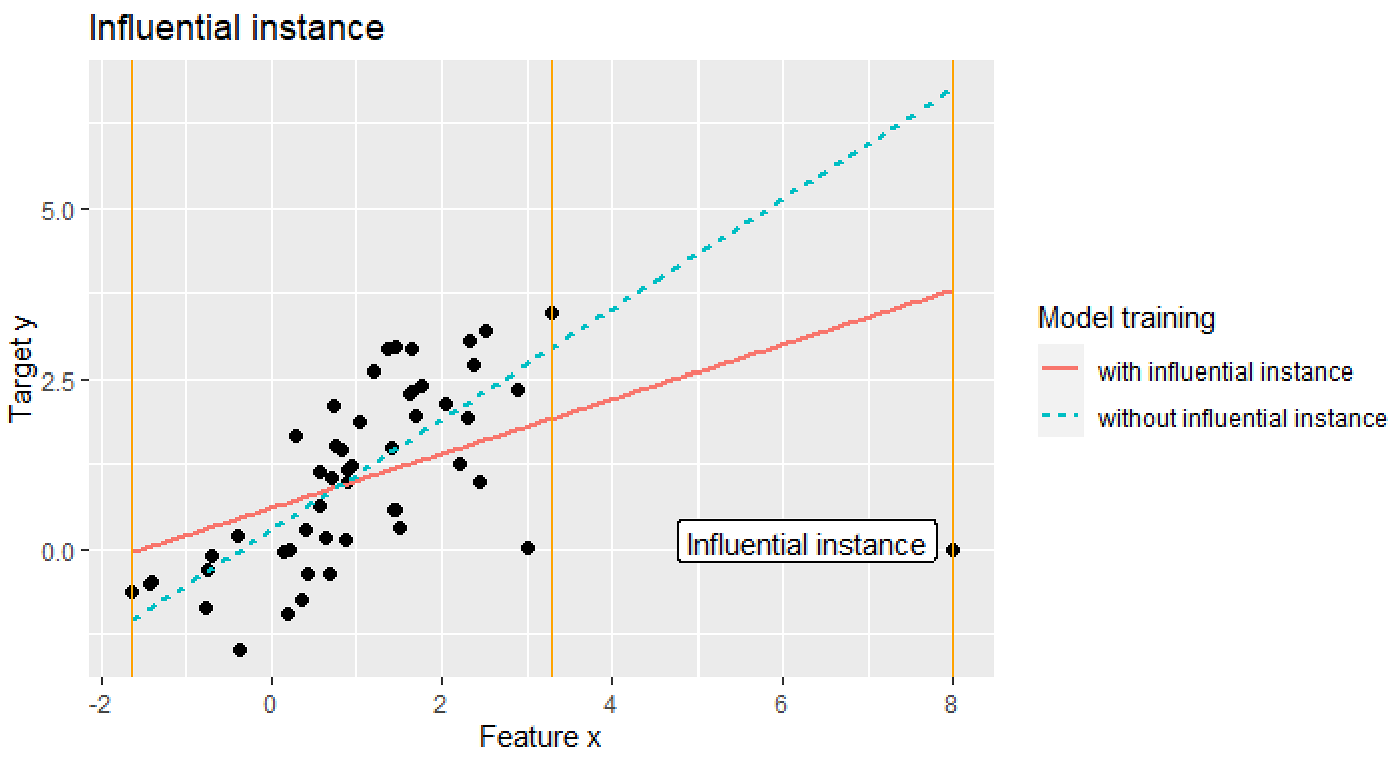


Figure 6 influential instance and top influential instance affected instances (orange line)

In figure 6, the orange line go through the top instances that affected by the influential instance the most (despite the one on the right, because that's the influential instance itself), they cut through the largest distance in between blue and red (the most difference).

This is only one-dimensional feature, if we have many features, by analyses the features of the influential instance we can use it to explain the datapoints highly influenced by it.

Influence Functions

Train the original data, delete one instance, train again, this seems to be time consuming, is there an easier way? The answer is yes, we can use influence Functions, which is a robust statistical method to measure how an influential instance can affect the parameters and prediction (Koh and Liang 2017 as cited in Molnar 2020). However, it only works with models like parameter models, like logistic regression, neural network, support vector machines.

Parameter vector after upweighting z   training data   Model parameter vector

$$\hat{\theta}_{\epsilon,z} = \arg\min_{\theta\in\Theta}(1-\epsilon)\frac{1}{n}\sum_{i=1}^{n}L(z_i,\theta)+\epsilon L(z,\theta)$$

The model's loss function   training instance we want to upweight

Formular 1 parameter for combined loss

The core idea of the influence function is to upweight one instance, and down weight the other instances to see what new the parameters look like on the combined loss function (formular 1).

To easier understand, we break down the function:

$L(z, \theta)$ is the loss for the instance we want to upweight to simulate its removal, we give it weight ϵ.

$\frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$ is the loss for the rest instances, we give it a weight $(1 - \epsilon)$.

Then we obtain the new parameters $\hat{\theta}_{\epsilon,z} = \arg\min_{\theta\in\Theta}$ by minimize the combined the loss.

After we got the new parameters, we can use the first and second derivative to get the following function for the influence of upweighting instance Z:

$$I_{\text{up,params}}(z) = \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_{\theta}L(z, \hat{\theta})$$

Formular 2 Influence calculation

$\nabla_{\theta}L(z, \hat{\theta})$ is the loss gradient with respect to the parameters for the upweighted instance z, it shows how the loss changes.

$H_{\hat{\theta}}^{-1}$ is the inverse Hessian matrix (second derivative of the loss respect to $\hat{\theta}$, it shows curvature of the loss). The reason we use gradient is that we cannot know (or too complex to calculate) the change of the parameters after upweighting an instance. By use gradient we can proximate the change:

$$\hat{\theta}_{-z} \approx \hat{\theta} - \frac{1}{n}I_{\text{up,params}}(z)$$

We use the example from Molnar (2020) to demonstrate the proximation.

```
x = seq(from = -1.2, to = 1.2, length.out = 100)     # parameter domain
y.fun = function(x) {                               # loss function
  -x - 2*x^2 - x^3 + 2 * x^4
}
y = y.fun(x)
expansion = function(x, x0 = 0) {
  d1 = function(x) -1 - 2*x - 3 * x^2 +  8 * x^3
  d2 = function(x)     - 2   - 6 * x   + 24 * x^2
  y.fun(x0) + d1(x0) * (x - x0) + 0.5 * (x - x0)^2*d2(x0)
}

dat = data.frame(x=x, y=y)
dat$type = "Actual loss"
dat2 = dat
dat2$type = "Quadratic expansion"
dat2$y = expansion(x)
dat3 = rbind(dat, dat2)

#pts  = data.frame(x = c(0, 2/6))

ggplot(dat3) +
  geom_line(aes(x = x, y = y, group = type, color = type, linetype = type)) +
  geom_vline(xintercept = 0, linetype = 2) +
  geom_vline(xintercept = 1/2, linetype = 2) +
  scale_y_continuous("Loss with upweighted instance z", labels = NULL, breaks = NULL) +
  scale_x_continuous("Model parameter", labels = NULL, breaks = NULL) +

  geom_point(x = 0, y = expansion(x = 0)) +
  geom_label(x = 0, label = expression(hat(theta)), y  = expansion(x=0), vjust = 2) +

  geom_point(x = 1/2, y = expansion(x = 1/2)) +
  geom_label(x = 1/2, y = expansion(x = 1/2), label = expression(hat(theta)[-z]), vjust = 2) +
```

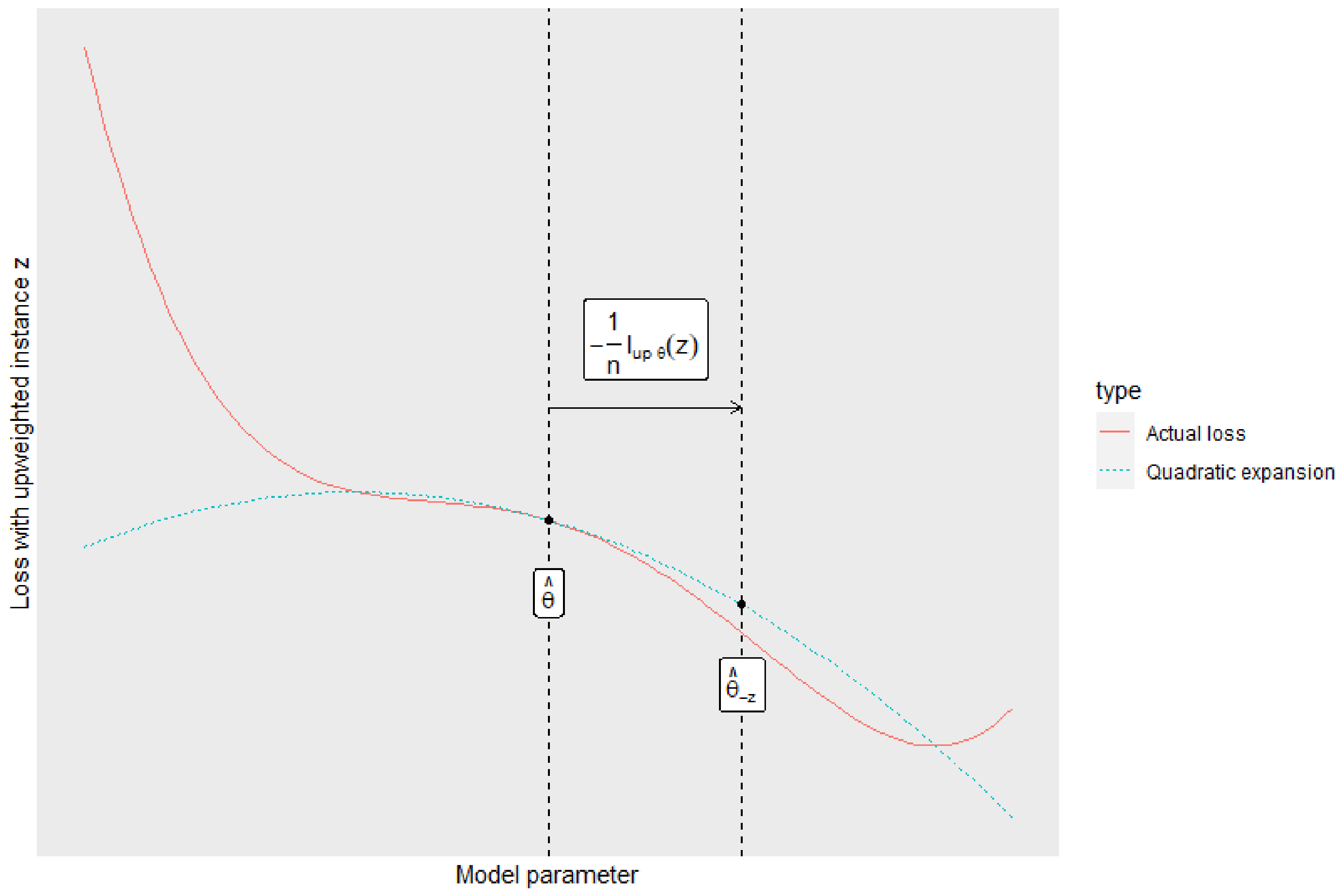


Figure 7 use quadratic expansion to proximate weight change

Figure 7 shows the quadratic expansion proximate the loss changes of deleting z instance. We can see only little difference between red and blue line when parameter change to $\hat{\theta}_{-z}$.

Boston data example

In the following section, we will demonstrate how to identify influential instances in Boston dataset.

We use Random Forest to train the model.

```
#from sklearn.neural_network import MLPRegressor
#nn = MLPRegressor(solver='lbfgs', hidden_layer_sizes=(8,8), random_state=0, max_iter=1000)
import xgboost
rf = xgboost.XGBRegressor()
rf.fit(train, labels_train)
rf2 = xgboost.XGBRegressor()
```

```
[00:33:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
rf.score(test, labels_test)
```

```
0.9009088645475026
```

```
MAE = np.mean(abs(rf.predict(test) - labels_test))
print('Random Forest Mean abusolute error', MAE)
```

```
Random Forest Mean abusolute error 1.9809417677860635
```

The Mean absolute error is 1.98.

Then we use z score to find out outliers, put the index of outliers in a list.

```
from scipy import stats
outlier_bool = ((np.abs(stats.zscore(train)) < 3).all(axis=1)) == 0     # (zscore < 3) == 0 means false, means outliers
```

```
outlier_index = [i for i, x in enumerate(outlier_bool) if x]
```

Then we pop out each outliers to test the influence, find the instance with the maximum influence.

```
influential_index = 0
max_influence = 0
for i in outlier_index:
  train_without = train.tolist()
  train_without.pop(i)

  labels_train_without = labels_train.tolist()
  labels_train_without.pop(i)

  model_without = rf2.fit(train_without, labels_train_without)

  influence = np.mean(abs(rf.predict(test) - model_without.predict(test)))
  print('influence of instance', i, ': ', influence)
  print('Score: ', model_without.score(test,labels_test))
  temp_influence = influence
  if max_influence < temp_influence:
    max_influence = temp_influence
    influential_index = i
print('Influential instance is instance: ', influential_index, '. Influence is:', max_influence)
```

```
Influential instance is instance:  26 . Influence is: 1.0456332
```

The result is instance 26 is the influential instance in the dataset.

Benefit of analyzing Influential instance

- **Influential instance** can help us understand the model's behavior (different model use different strategy to predict.
- It can also help us find the flaw in the model. By analyses the influential instance, we can find out whether a model is biased towards a group of population.
- Sometimes, it's hard to check all records in the data to find if there are errors, instead we can just check a few instances, influential instances are much better choice than instances selected by other method such as random selection (Koh and Liang 2017 as cited in Molnar 2020)

Drawbacks of influential instance

- Deletion diagnostics method are expensive to calculate, Influence functions on the other hand are just approximate.
- We only measure the impact of removing a single instance, if there are impact from the interactions of multiple instances, we cannot deal with that.

Conclusion

In this review we discussed 4 example-based method to interpret a machine leaning model. For local explanation method, we can use counterfactual examples, for global explanation, we can use influential instances. Prototypes and criticisms can use in both situations. We will show summaries them with images below:

Counterfactual explanation (shown in figure below) is we offer a counter example of the original instance, which lead to a different prediction, to explain the model.

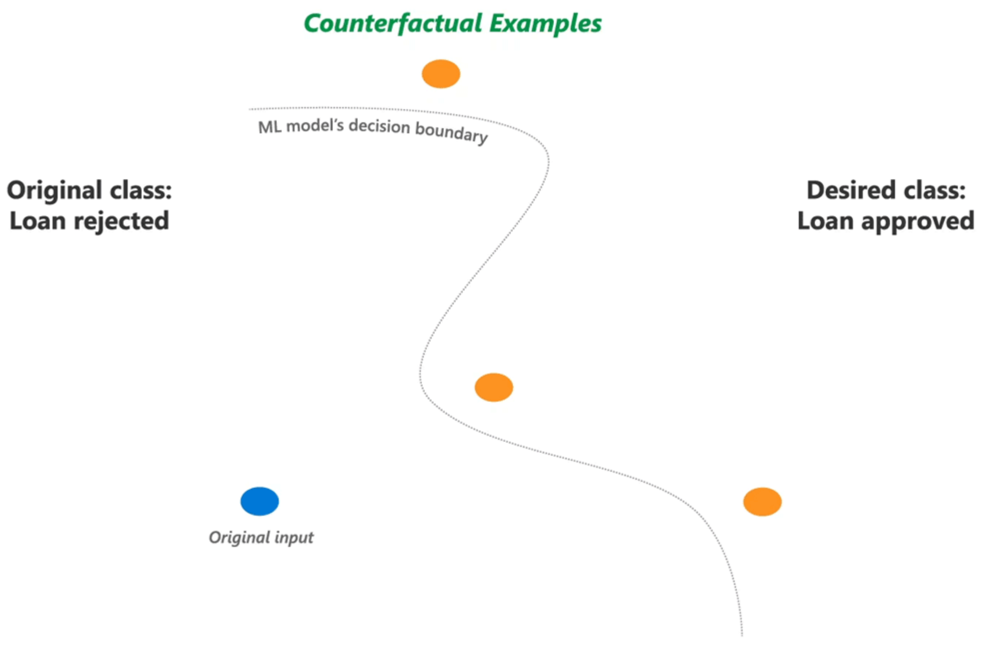


Figure 8 Counterfactual Examples (Image from Sharma 2020)

The **Adversarial examples** is the examples that added to the data that make the model make wrong decisions, we discussed how it happens, and how to defense an adversarial example attack.

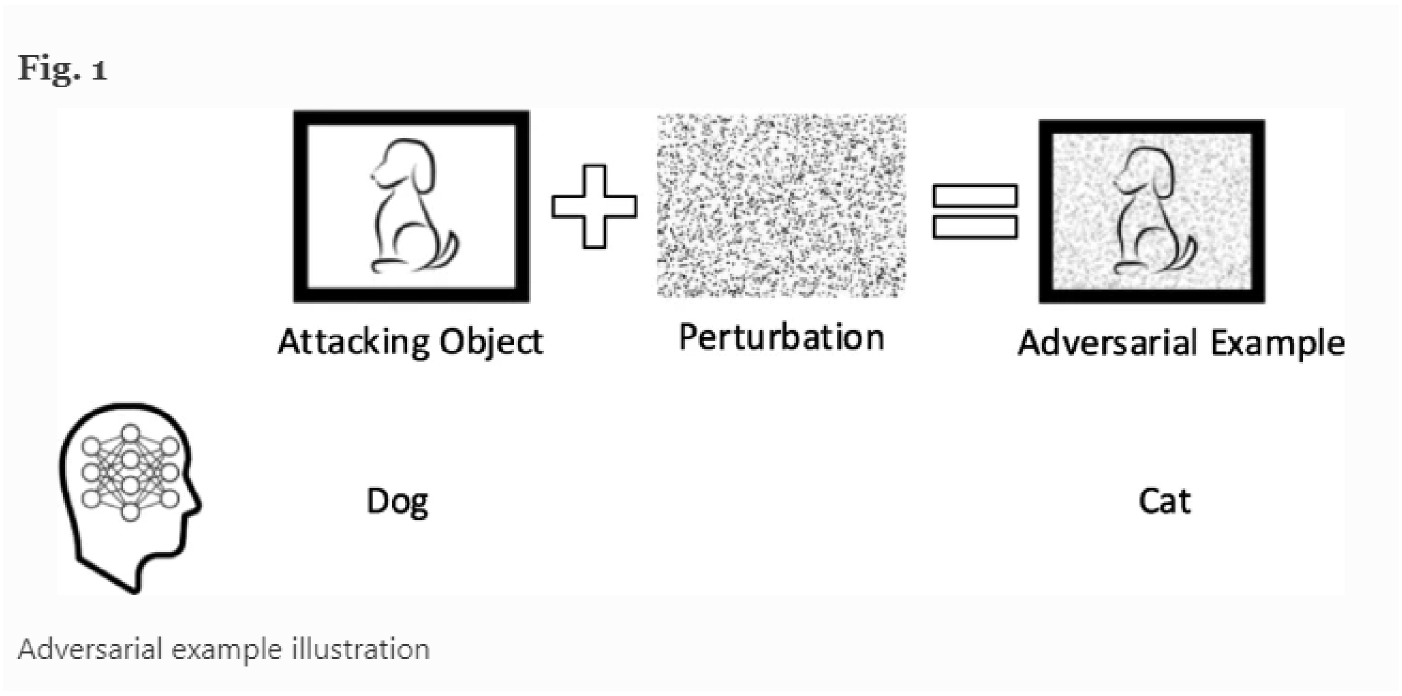


Figure 9 Adversarial examples (Image from Sun, Tan, and Zhou 2018)

Prototypes are the presentative of all the data, **criticism** is the datapoint which cannot represent by prototypes. Since they are complementing of each other, so we need analyses both to better interpret the data.

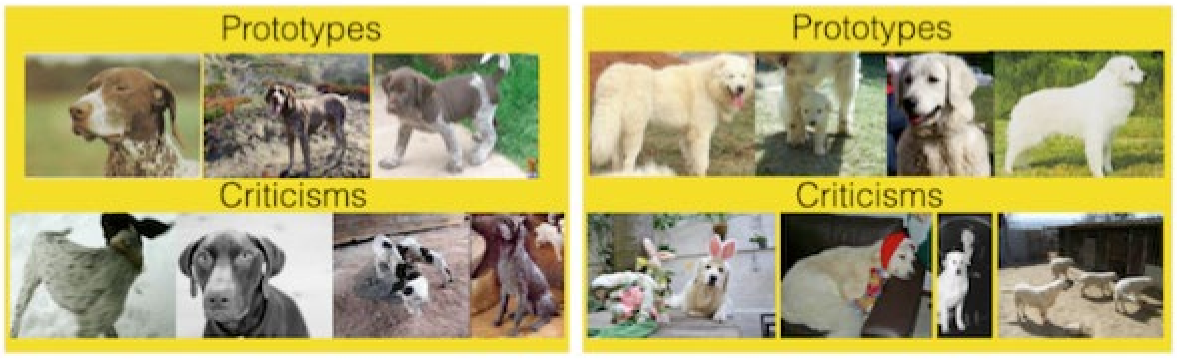


Figure 10 Prototypes and criticisms for two types of dog breeds from the ImageNet dataset (Molnar, C 2020). Phototypes are usually showing the face of the dog, which critics usually show other non-important features such as the back of the dog, grey color picture, costume of the dog wearing).

Influential instance can help us understand the model’s behavior and fix error in the data.

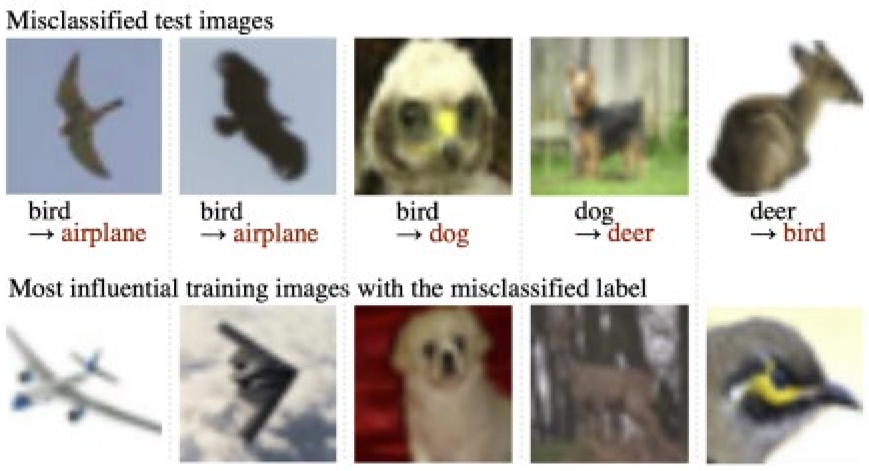

Misclassified test images
bird → airplane
bird → airplane
bird → dog
dog → deer
deer → bird
Most influential training images with the misclassified label

Figure 11 misclassified image and most influential training images with the misclassified label (image from Kobayashi et al.2020).

Reference

Molnar, C 2020, *Interpretable Machine Learning A Guide for Making Black Box Models Explainable,* Leanpub, Leanpup.com.

Verma, S., Boonsanong, V, Hoang, M, Dickerson, J, Shah, C and Hines, K E, 2020. 'Counterfactual explanations for machine learning: A review'. *arXiv preprint arXiv:*2010.10596.

David Lewis. 1973. *Counterfactuals*. Blackwell Publishers, Oxford.

Szegedy, C, Zaremba, W, Sutskever, I, Brunna, J, Erhan, D, Goodfellow, I, Fergus, R, 2013, "Intriguing properties of neural networks." *arXiv preprint arXiv*:1312.6199.

Goodfellow, I J, Shlens J, and Szegedy ,C. 2014, "Explaining and harnessing adversarial examples." *arXiv*

preprint arXiv:1412.6572.

Tsui, K, 2018, *Perhaps the Simplest Introduction of Adversarial Examples Ever,* Towards Data Science, viewed 6 December 2022. < https://towardsdatascience.com/perhaps-the-simplest-introduction-of-adversarial-examples-ever-c0839a759b8d#:~:text=Fast%20Gradient%20Sign%20Method%20(FGSM)&text=In%20essence%2C%20FGSM%20is%20to,small%20number%20via%20max%20norm.>.

Kim, B, Khanna, R and Koyejo, O.O, 2016. Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems*, *29*.

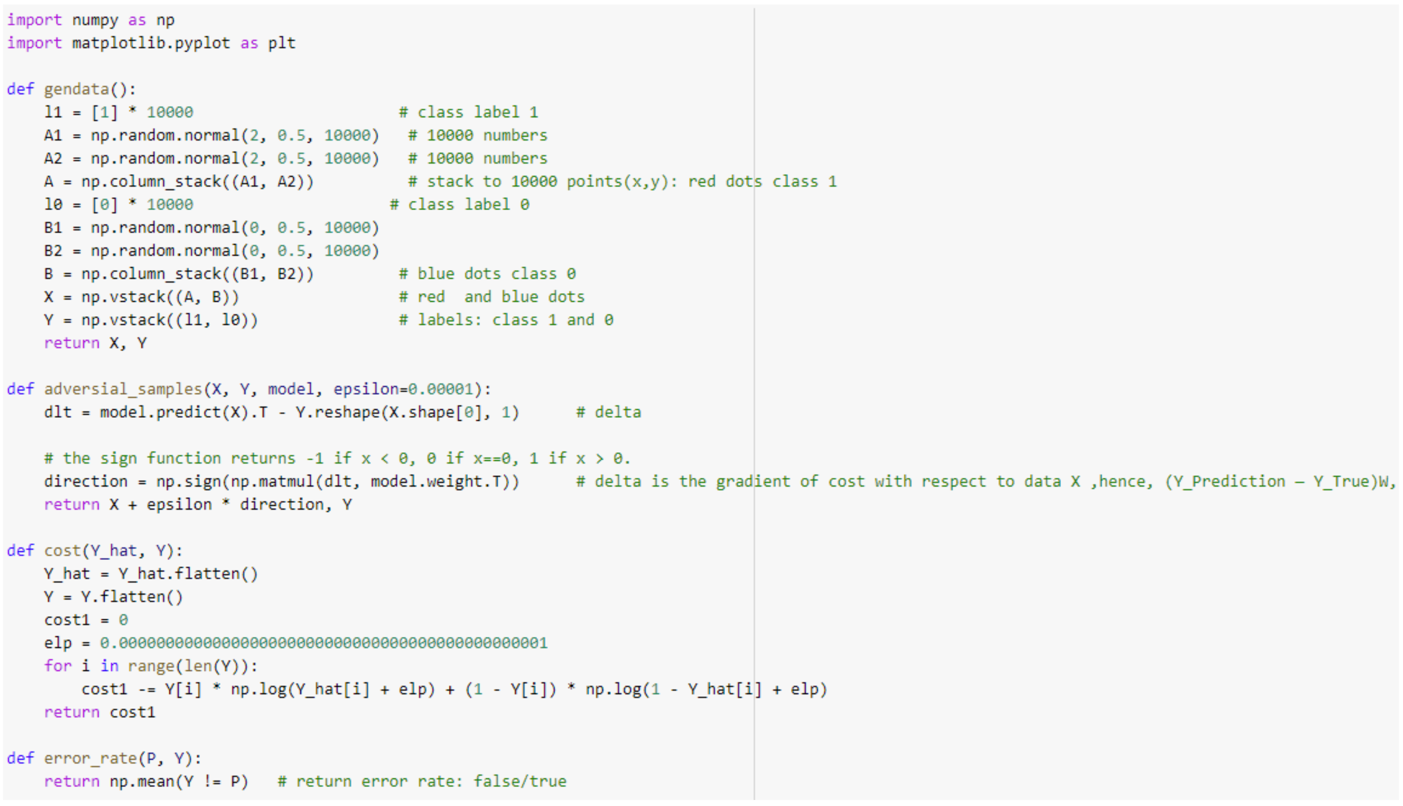
Shama A, 2020, *Open-source library provides explanation for machine learning through diverse counterfactuals*, Microsof, viewed 12 December, https://www.microsoft.com/en-us/research/blog/open-source-library-provides-explanation-for-machine-learning-through-diverse-counterfactuals/.

Sun, L., Tan, M. and Zhou, Z., 2018. ‘A survey of practical adversarial example attacks’. *Cybersecurity*, *1*(1), pp.1-9.

Kobayashi, S, Yokoi, S, Suzuki, J. and Inui, K, 2020. ‘Efficient estimation of influence of a training instance’. *arXiv preprint arXiv:2012.04207*.

Appendix

A:



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

def gendata():
    l1 = [1] * 10000                      # class label 1
    A1 = np.random.normal(2, 0.5, 10000)   # 10000 numbers
    A2 = np.random.normal(2, 0.5, 10000)   # 10000 numbers
    A = np.column_stack((A1, A2))          # stack to 10000 points(x,y): red dots class 1
    l0 = [0] * 10000                      # class label 0
    B1 = np.random.normal(0, 0.5, 10000)
    B2 = np.random.normal(0, 0.5, 10000)
    B = np.column_stack((B1, B2))         # blue dots class 0
    X = np.vstack((A, B))                 # red  and blue dots
    Y = np.vstack((l1, l0))               # labels: class 1 and 0
    return X, Y

def adversial_samples(X, Y, model, epsilon=0.00001):
    dlt = model.predict(X).T - Y.reshape(X.shape[0], 1)      # delta

    # the sign function returns -1 if x < 0, 0 if x==0, 1 if x > 0.
    direction = np.sign(np.matmul(dlt, model.weight.T))      # delta is the gradient of cost with respect to data X ,hence, (Y_Prediction – Y_True)W,
    return X + epsilon * direction, Y

def cost(Y_hat, Y):
    Y_hat = Y_hat.flatten()
    Y = Y.flatten()
    cost1 = 0
    elp = 0.000000000000000000000000000000000000000000001
    for i in range(len(Y)):
        cost1 -= Y[i] * np.log(Y_hat[i] + elp) + (1 - Y[i]) * np.log(1 - Y_hat[i] + elp)
    return cost1

def error_rate(P, Y):
    return np.mean(Y != P)   # return error rate: false/true
```

```
class logistic_regression(object):
    def fit(self, X, Y, learning_rate=0.0000003, epoch=1000):
        X = np.array(X, dtype="float32")
        Y = np.array(Y, dtype="float32")
        #
        N, D = X.shape
        Y = Y.reshape(N, 1)
        #
        dlt = np.zeros([N, 1], dtype="float32")
        dW = np.zeros([1, D], dtype="float32")
        db = 0
        self.weight = np.zeros([D, 1], dtype="float32")
        self.beta = 0
        #
        c = []
        #
        for n in range(epoch):
            dlt = self.predict(X).T - Y
            dW = np.matmul(dlt.T, X).T
            db = dlt.sum() / N
            #
            self.weight -= learning_rate * dW            # get optimal weight
            self.beta -= learning_rate * db              # get optimal beta
            if n % 1000 == 0:                    # print detail on every 1000 iteration
                c_new = cost(self.predict(X).T, Y)
                c.append(c_new)
                err = error_rate(self.predict_class(X).T, Y)
                print("epoch:", n, "cost:", c_new, "error rate:", err)

    def predict(self, X):
        z = np.matmul(self.weight.T, X.T) + self.beta
        return 1 / (1 + np.exp(-z))       # output is between 0 and 1

    def predict_class(self, X):
        predictclass = self.predict(X)
        return (predictclass >= 0.5) * 1
```

```
X, Y = gendata()
model = logistic_regression()
model.fit(X, Y, learning_rate=0.0005, epoch=20000)
print('Error Rate without adversarial examples:', error_rate(model.predict_class(X).T, Y.reshape(X.shape[0], 1)))
Xadv, Y = adversial_samples(X, Y, model, epsilon=0.2)
print('Error Rate with adversarial examples, epsilon = 0.2:', error_rate(model.predict_class(Xadv).T, Y.reshape(Xadv.shape[0], 1)))
Xadv2, Y = adversial_samples(X, Y, model, epsilon=0.5)
print('Error Rate with adversarial examples, epsilon = 0.5:', error_rate(model.predict_class(Xadv2).T, Y.reshape(Xadv2.shape[0], 1)))
```

```
plt.scatter(X[10000:, 0], X[10000:, 1], color='red', alpha=0.4, label='Original Data')
plt.scatter(X[:9999, 0], X[:9999, 1], color='blue', alpha=0.4, label='Original Data')
```

```
plt.scatter(X[:9999, 0], X[:9999, 1], color='blue', alpha=0.4, label='Original Data')
plt.scatter(Xadv[:9999, 0], Xadv[:9999, 1], color='green', alpha=0.4, label='Adversarial Examples, Epsilon = 0.2')
plt.scatter(Xadv2[:9999, 0], Xadv2[:9999, 1], color='purple', alpha=0.4, label='Adversarial Examples, Epsilon = 0.5')
plt.scatter(X[10000:, 0], X[10000:, 1], color='red', alpha=0.4, label='Original Data')
plt.scatter(Xadv[10000:, 0], Xadv[10000:, 1], color='orange', alpha=0.4, label='Adversarial Examples, Epsilon = 0.2')
plt.scatter(Xadv2[10000:, 0], Xadv2[10000:, 1], color='yellow', alpha=0.4, label='Adversarial Examples, Epsilon = 0.5')
plt.legend(loc='best')
plt.show()
```

B:

```r
library(ggplot2)
library(dplyr)

## first graph - original data points
set.seed(1)
dat1 = data.frame(x1 = rnorm(20, mean = 4, sd = 0.3), x2 = rnorm(20, mean = 1, sd = 0.3))
dat2 = data.frame(x1 = rnorm(30, mean = 2, sd = 0.2), x2 = rnorm(30, mean = 2, sd = 0.2))
dat3 = data.frame(x1 = rnorm(40, mean = 3, sd = 0.2), x2 = rnorm(40, mean = 3))
dat4 = data.frame(x1 = rnorm(7, mean = 4, sd = 0.1), x2 = rnorm(7, mean = 2.5, sd = 0.1))

dat = rbind(dat1, dat2, dat3, dat4)
dat$type1 = "data"
dat$type1[c(7, 23, 77)] = "prototype"
dat$type1[c(81,95)] = "criticism"

ggplot(dat, aes(x = x1, y = x2)) + geom_point(alpha = 0.7) +
  geom_point(data = filter(dat, type1 !='data'), aes(shape = type1), size = 9, alpha = 1, color = "blue") +
  scale_shape_manual(breaks = c("prototype", "criticism"), values = c(18, 19))


## second graph - phototypes
set.seed(42)
n = 40
# create 4 groups of points from three gaussians in 2d
dt1 = data.frame(x1 = rnorm(n, mean = 1, sd = 0.1), x2 = rnorm(n, mean = 1, sd = 0.3))
dt2 = data.frame(x1 = rnorm(n, mean = 4, sd = 0.3), x2 = rnorm(n, mean = 2, sd = 0.3))
dt3 = data.frame(x1 = rnorm(n, mean = 3, sd = 0.5), x2 = rnorm(n, mean = 3, sd = 0.3))
dt4 = data.frame(x1 = rnorm(n, mean = 2.6, sd = 0.1), x2 = rnorm(n, mean = 1.7, sd = 0.1))
dt = rbind(dt1, dt2, dt3, dt4)


radial = function(x1, x2, sigma = 1) {
  dist = sum((x1 - x2)^2)        # why not use euclidean <- function(a,b) sqrt(sum((a-b)^2))
  exp(-dist/(2*sigma^2))
}
```

```r
cross.kernel = function(d1, d2) {
  kk = c()
  for (i in 1:nrow(d1)) {
    for (j in 1:nrow(d2)) {
      res = radial(d1[i,], d2[j,])
      kk = c(kk, res)
    }
  }
  mean(kk)
}

mmd2 = function(d1, d2) {
  cross.kernel(d1, d1) - 2 * cross.kernel(d1, d2) + cross.kernel(d2,d2)
}

# create 3 variants of prototypes
pt1 = rbind(dt1[c(1,2),], dt4[1,])  # 2 points from group 1, 1 points from group 4
pt2 = rbind(dt1[1,], dt2[3,], dt3[19,])  # 1 points from group 1, 1 points from group 2, 1 points from group 3
pt3 = rbind(dt2[3,], dt3[19,])  # 1 points from group 2, 1 points from group 3

# create plot with all data and density estimation
p = ggplot(dt, aes(x = x1, y = x2)) +
  stat_density_2d(geom = "tile", aes(fill = ..density..), contour = FALSE, alpha = 0.9) +
  geom_point() +
  scale_fill_gradient2(low = "white", high = "blue", guide = "none") +
  scale_x_continuous(limits = c(0, NA)) +
  scale_y_continuous(limits = c(0, NA))
# create plot for each prototype
p1 = p + geom_point(data = pt1, color = "red", size = 4) + geom_density_2d(data = pt1, color = "red") +
  ggtitle(sprintf("%.3f MMD2", mmd2(dt, pt1)))

p2 = p + geom_point(data = pt2, color = "red", size = 4) +
  geom_density_2d(data = pt2, color = "red") +
  ggtitle(sprintf("%.3f MMD2", mmd2(dt, pt2)))
```

```r
p3 = p + geom_point(data = pt3, color = "red", size = 4) +
  geom_density_2d(data = pt3, color = "red") +
  ggtitle(sprintf("%.3f MMD2", mmd2(dt, pt3)))
# TODO: Add custom legend for prototypes

# overlay mmd measure for each plot
gridExtra::grid.arrange(p, p1, p2, p3, ncol = 2)



# third plot - criticisms
witness = function(x, dist1, dist2, sigma = 1) {
  k1 = apply(dist1, 1, function(z) radial(x, z, sigma = sigma))
  k2 = apply(dist2, 1, function(z) radial(x, z, sigma = sigma))
  mean(k1) - mean(k2)
}

w.points.indices = c(125, 2, 60, 19, 100)
wit.points = dt[w.points.indices,]
wit.points$witness = apply(wit.points, 1, function(x) round(witness(x[c("x1", "x2")], dt, pt2, sigma = 1), 3))

p + geom_point(data = pt2, color = "red") +
  geom_density_2d(data = pt2, color = "red") +
  ggtitle(sprintf("%.3f MMD2", mmd2(dt, pt2))) +
  geom_label(data = wit.points, aes(label = witness), alpha = 0.9, vjust = "top") +
  geom_point(data = wit.points, color = "black", shape = 17, size = 4)
```