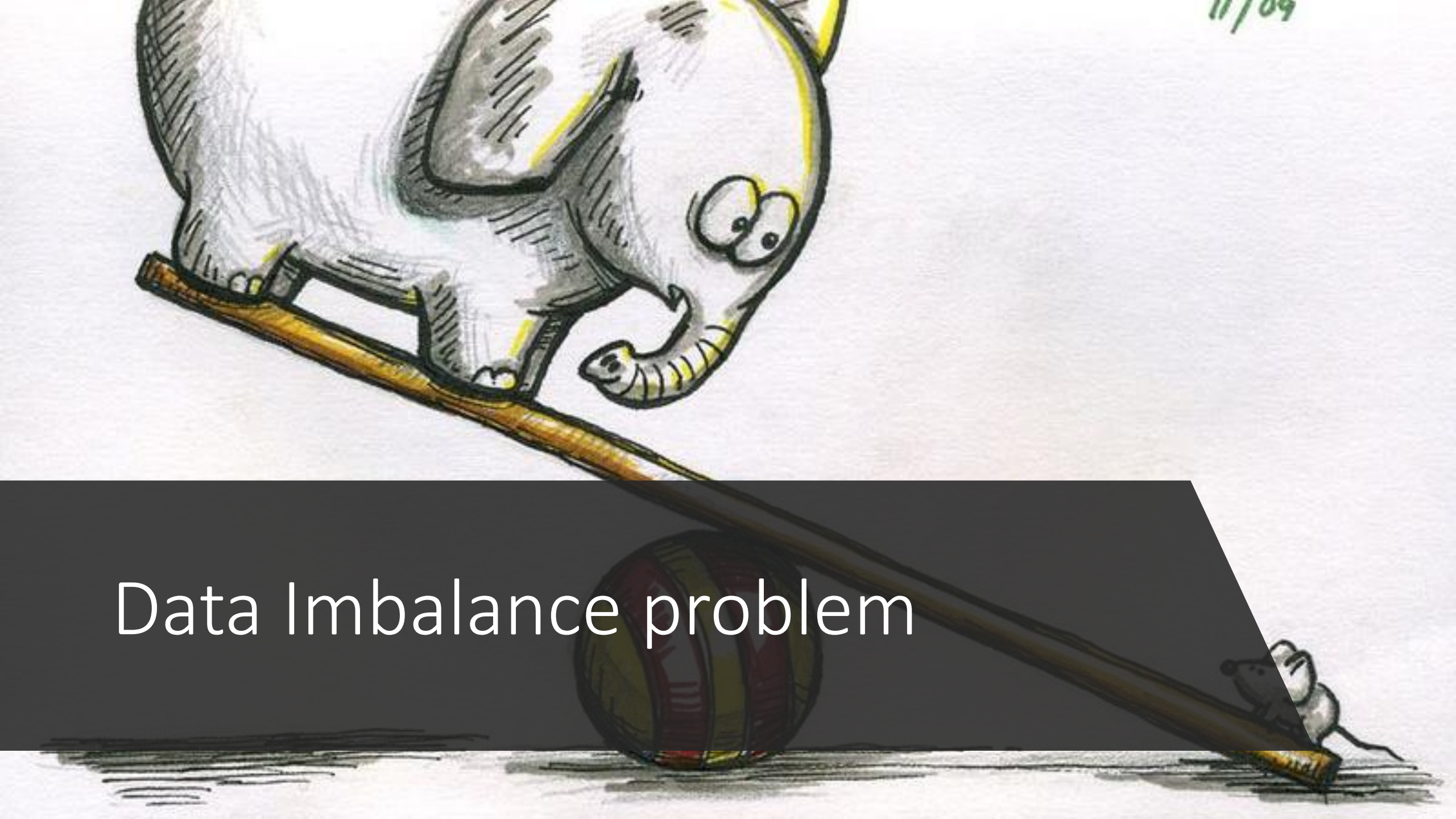
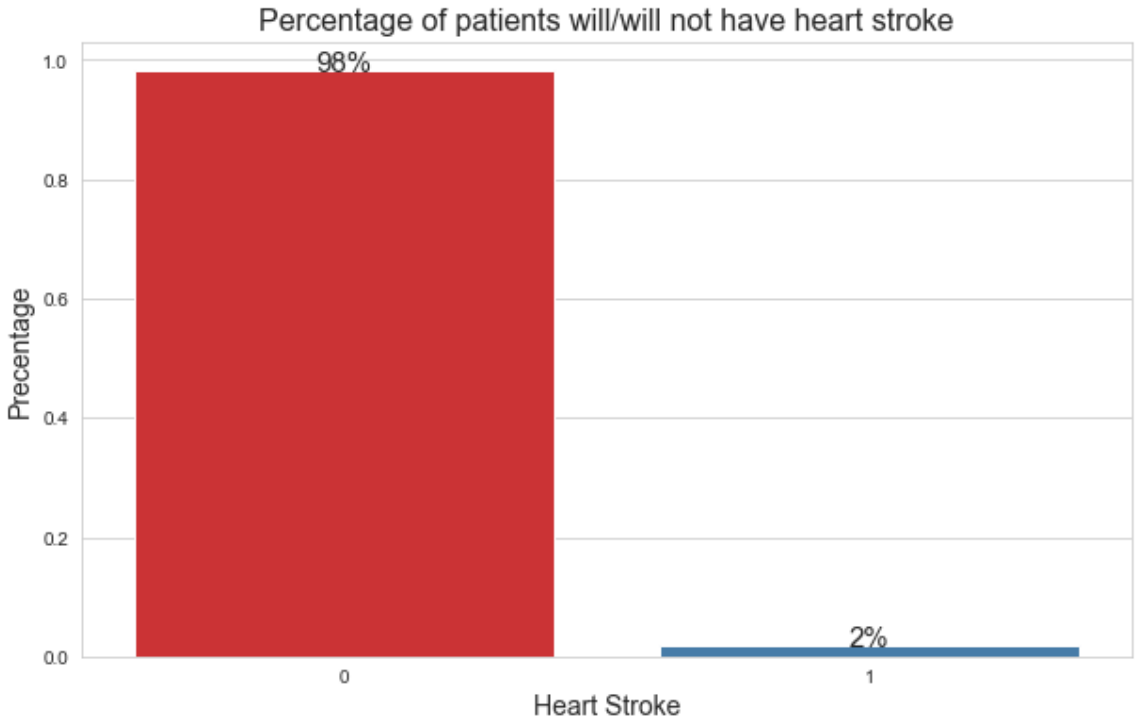


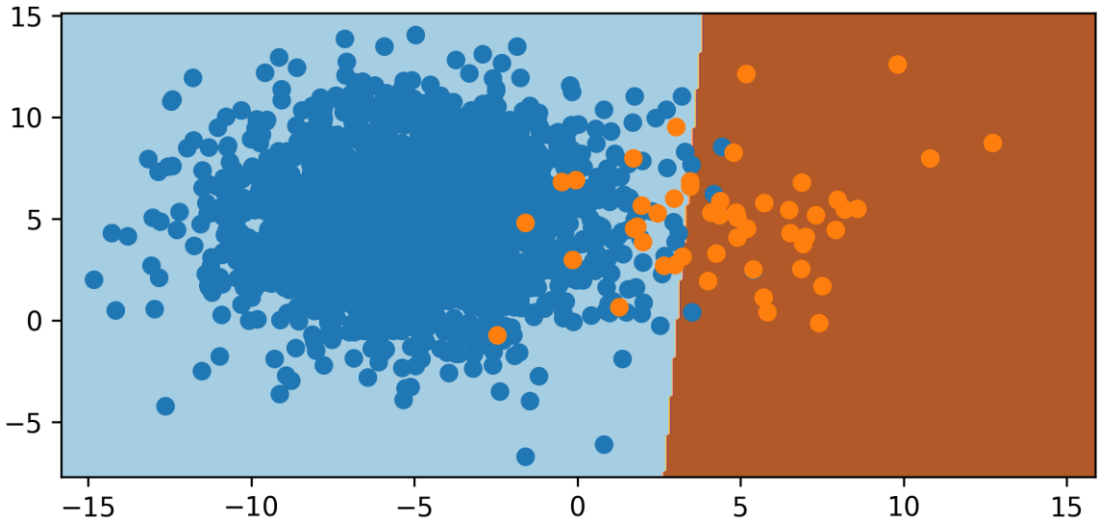
Data Imbalance problem

Is accuracy correct way to measure Quality of model?

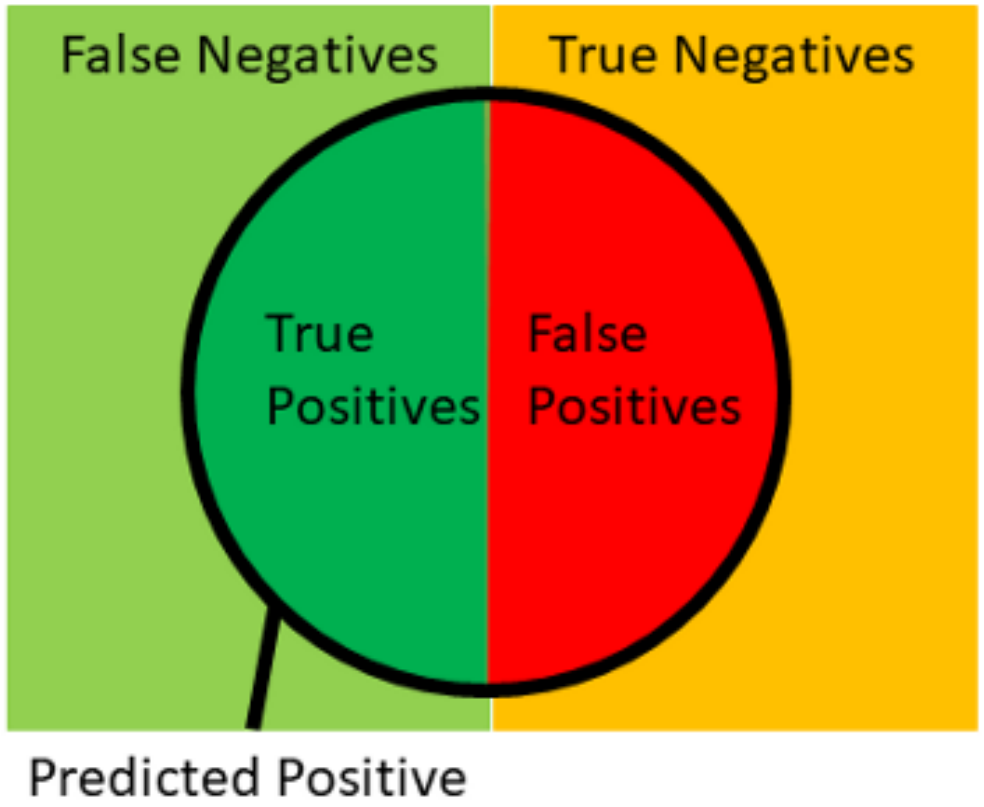
Why this happen?

- Fraud Detection
- Anomaly Detection
- Healthcare





Confusion Matrix



Confusion Matrix		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

The trick of Accuracy

Accuracy: 98%
F1 Score: 0

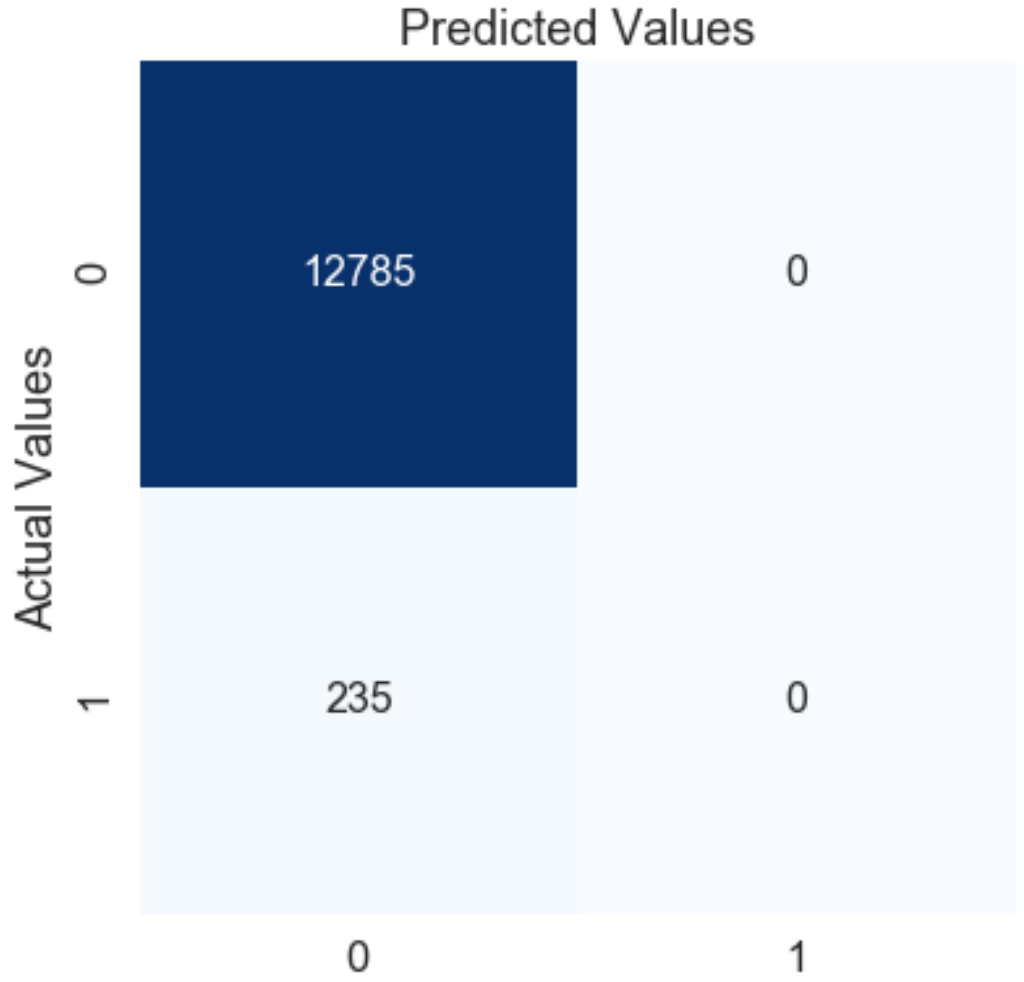
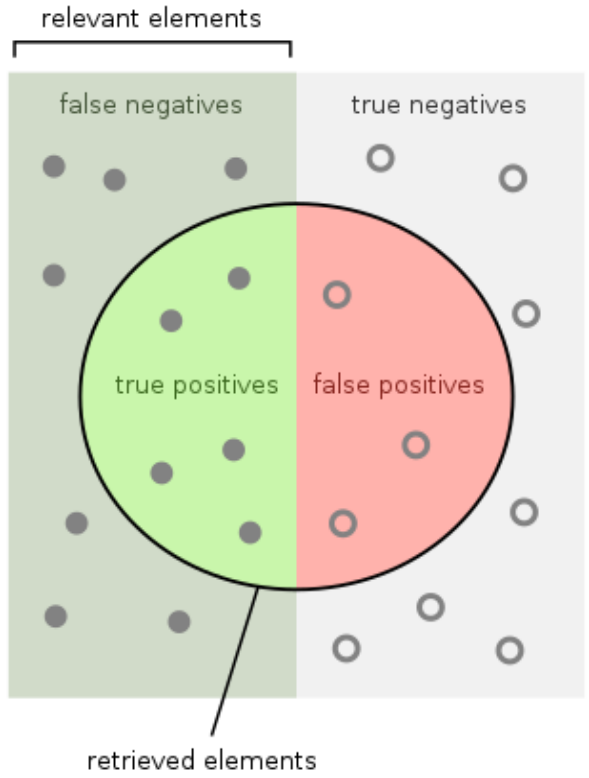


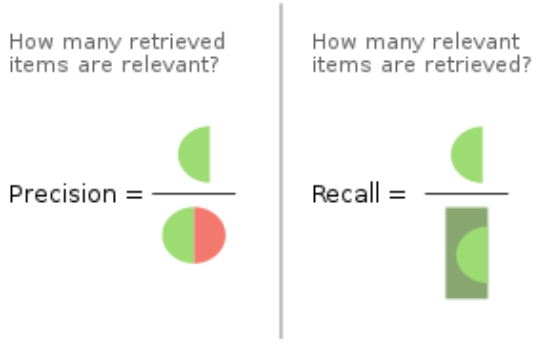
Image: Analytics Vidya

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

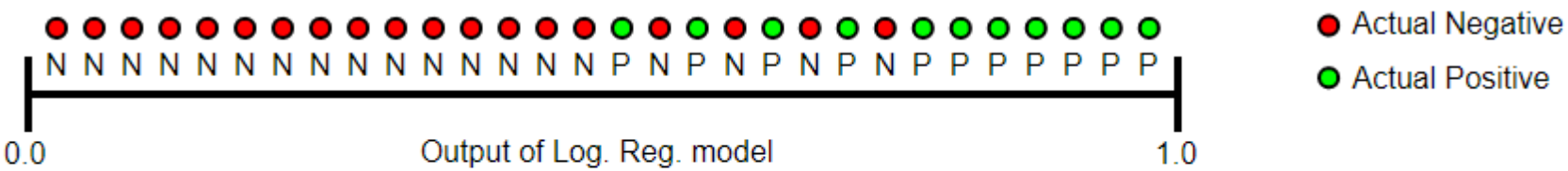
$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$





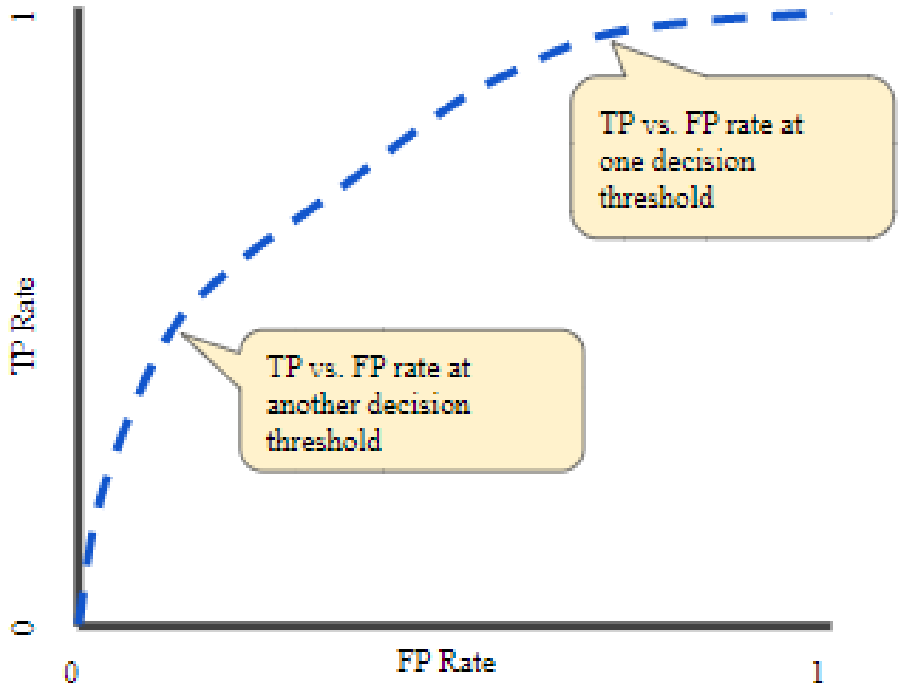
How to measure quality of model?

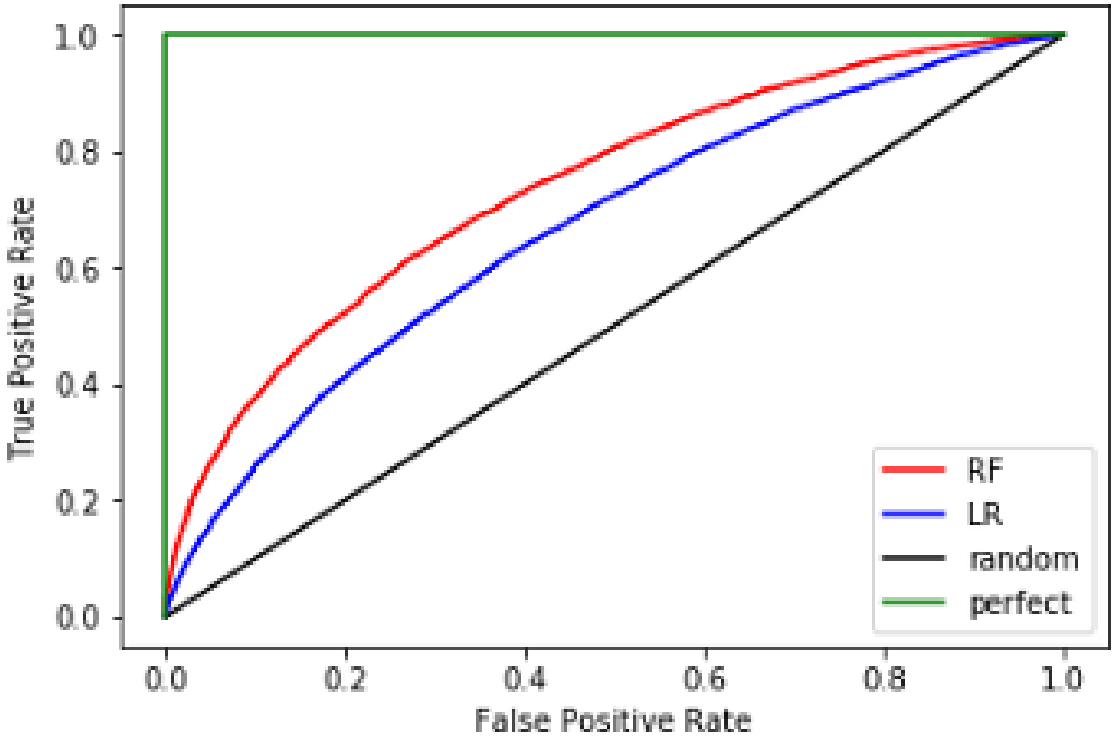


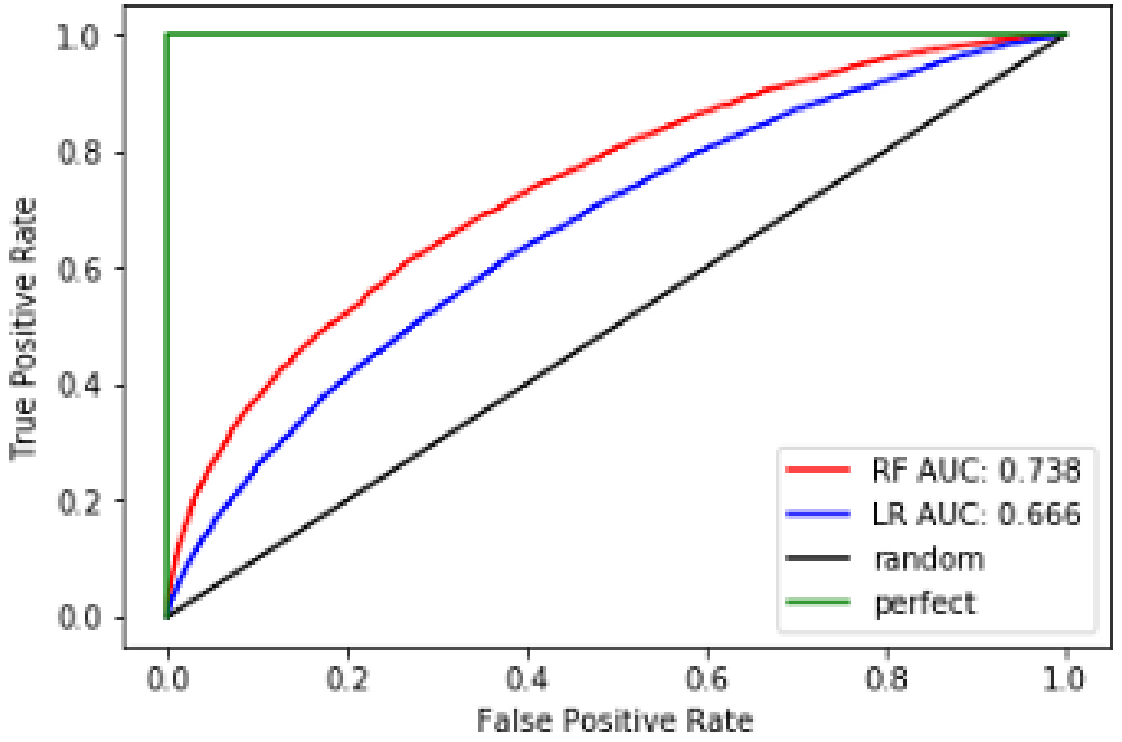
$$TPR = \frac{TP}{TP + FN}$$

TPR ↑, FPR ↑ and TPR ↓, FPR ↓

$$FPR = \frac{FP}{FP + TN}$$





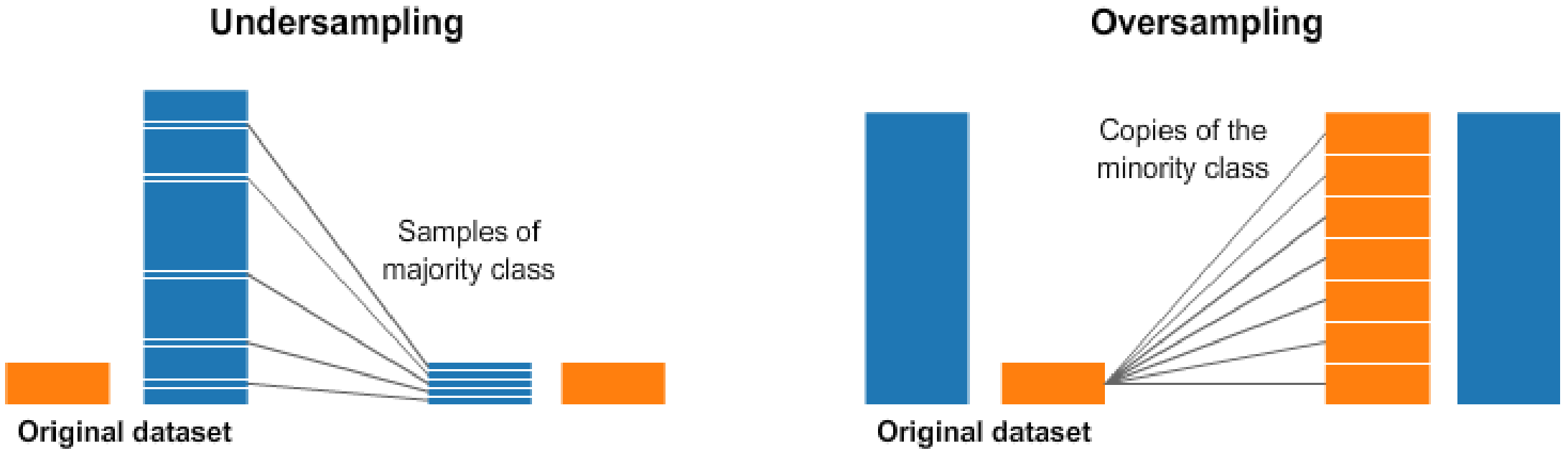


ROC Curve and ROC AUC Score

Methods to Overcome Data Imbalance Problem

- **Class weight**
- **Oversampling**
  - **Random oversampling**
  - **Synthetic Minority Over-sampling Technique (SMOTE)**
  - **ADASYN**
- **Undersampling**
  - **Random undersampling**
  - **Cluster centroids**
  - **Near miss**
  - **Tomeks links**

To the very core



Class weight

- Provide a weight for each class which places more emphasis on the minority classes

```
from sklearn.utils.class_weight import compute_class_weight
weights = compute_class_weight('balanced', classes, y)
```

wj=n_samples / (n_classes * n_samplesj)

Here,
•wj is the weight for each class(j signifies the class)
•n_samples is the total number of samples or rows in the dataset
•n_classes is the total number of unique classes in the target
•n_samplesj is the total number of rows of the respective class

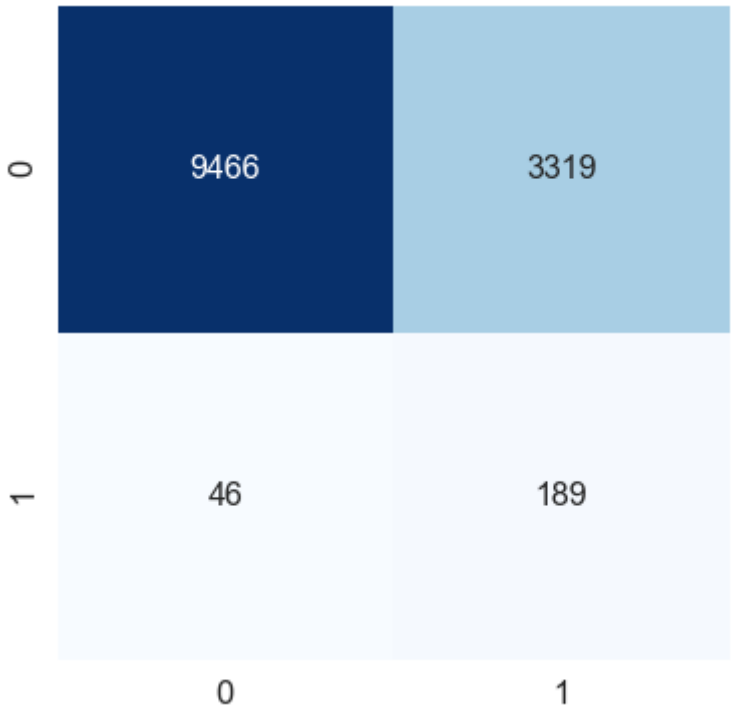
You modify the loss function to accommodate for Imbalanced data

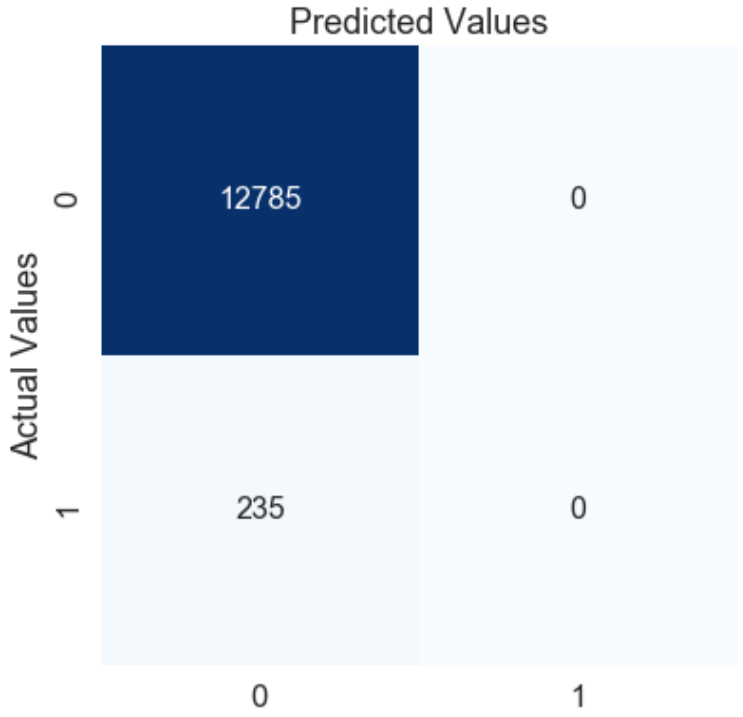
$$log\ loss = \frac{1}{N}\sum_{i=1}^{N}[-(y_i * log(\widehat{y_i}) + (1-y_i) * log(1-\widehat{y_i}))$$

$$log\ loss = \frac{1}{N}\sum_{i=1}^{N}[-(w_0(y_i * log(\widehat{y_i})) + w_1((1-y_i) * log(1-\widehat{y_i})))]$$

Image: Analytics Vidya

After Experimenting



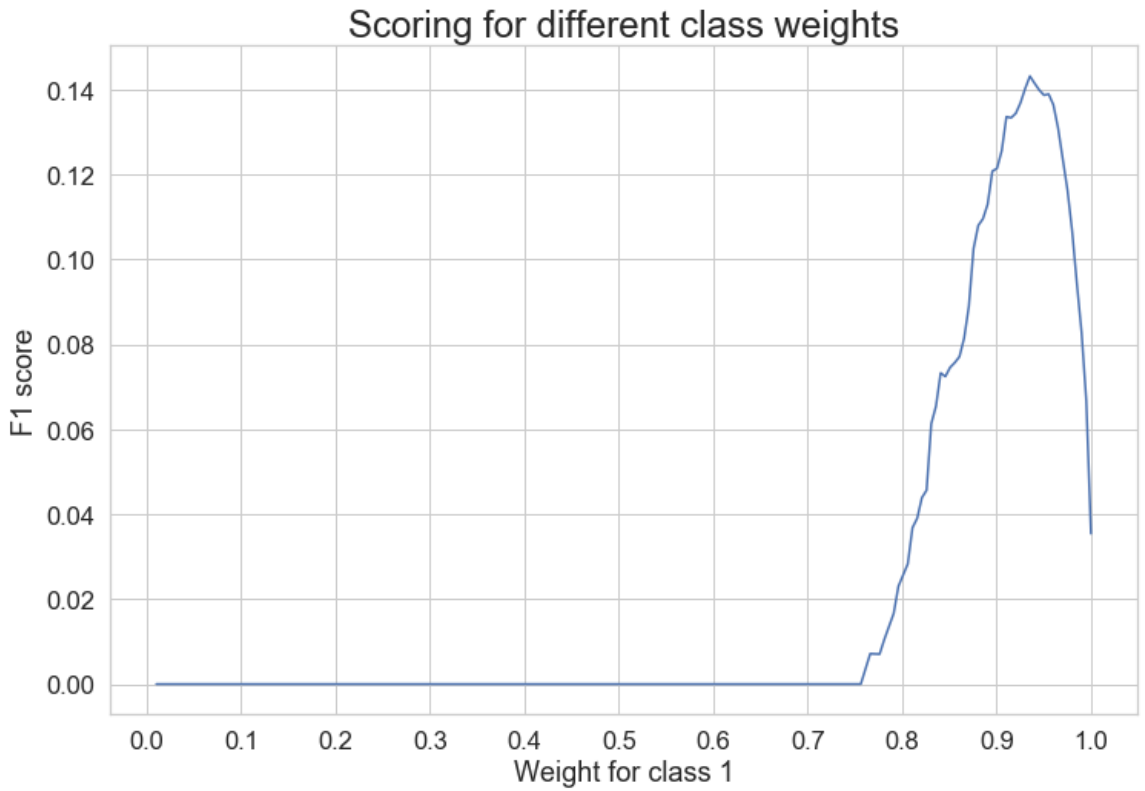


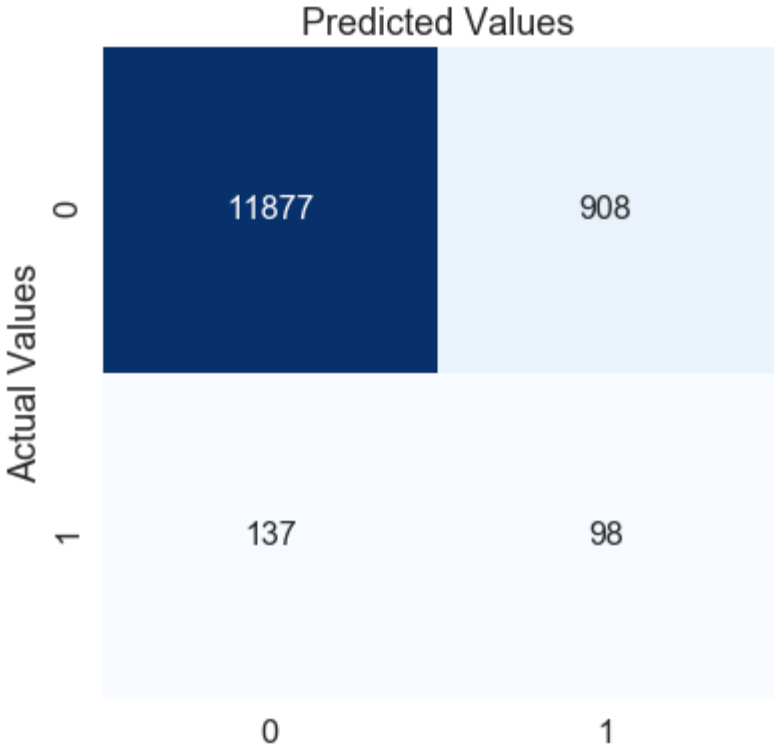
Before Using Class weights

The f1-score for the testing data: 0.10098851188885921

Image: Analytics Vidya

More Experimenting





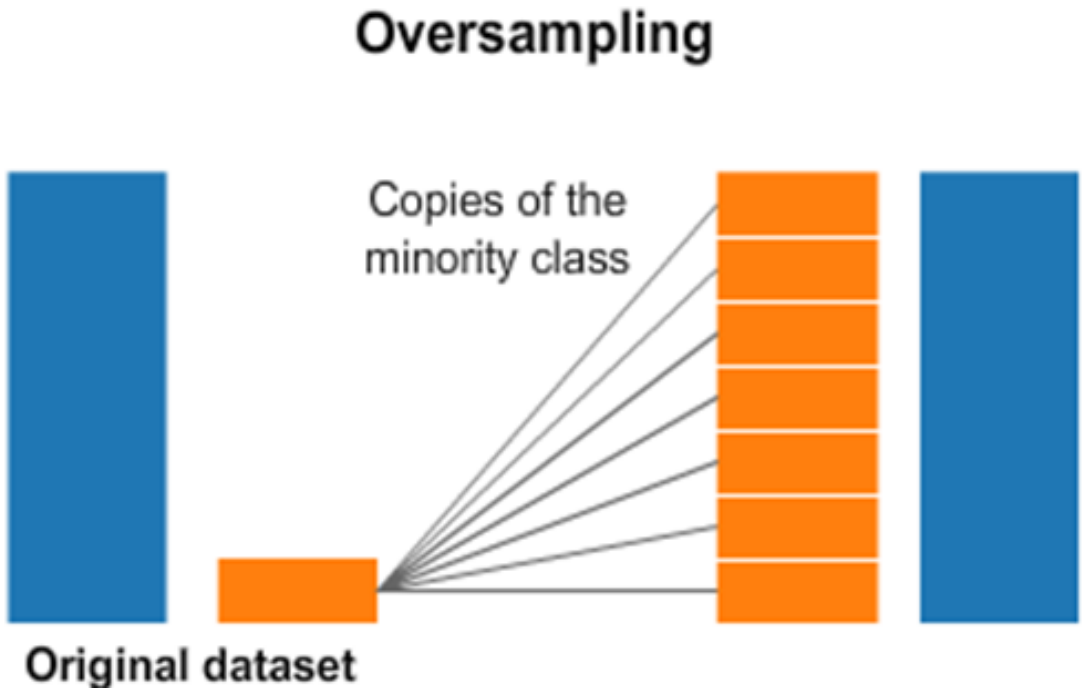
f1-score for the testing data: 0.157937147461724 4

Image: Analytics Vidya

Oversampling

- Oversampling the minority classes to increase the number of minority observations until we've reached a balanced dataset

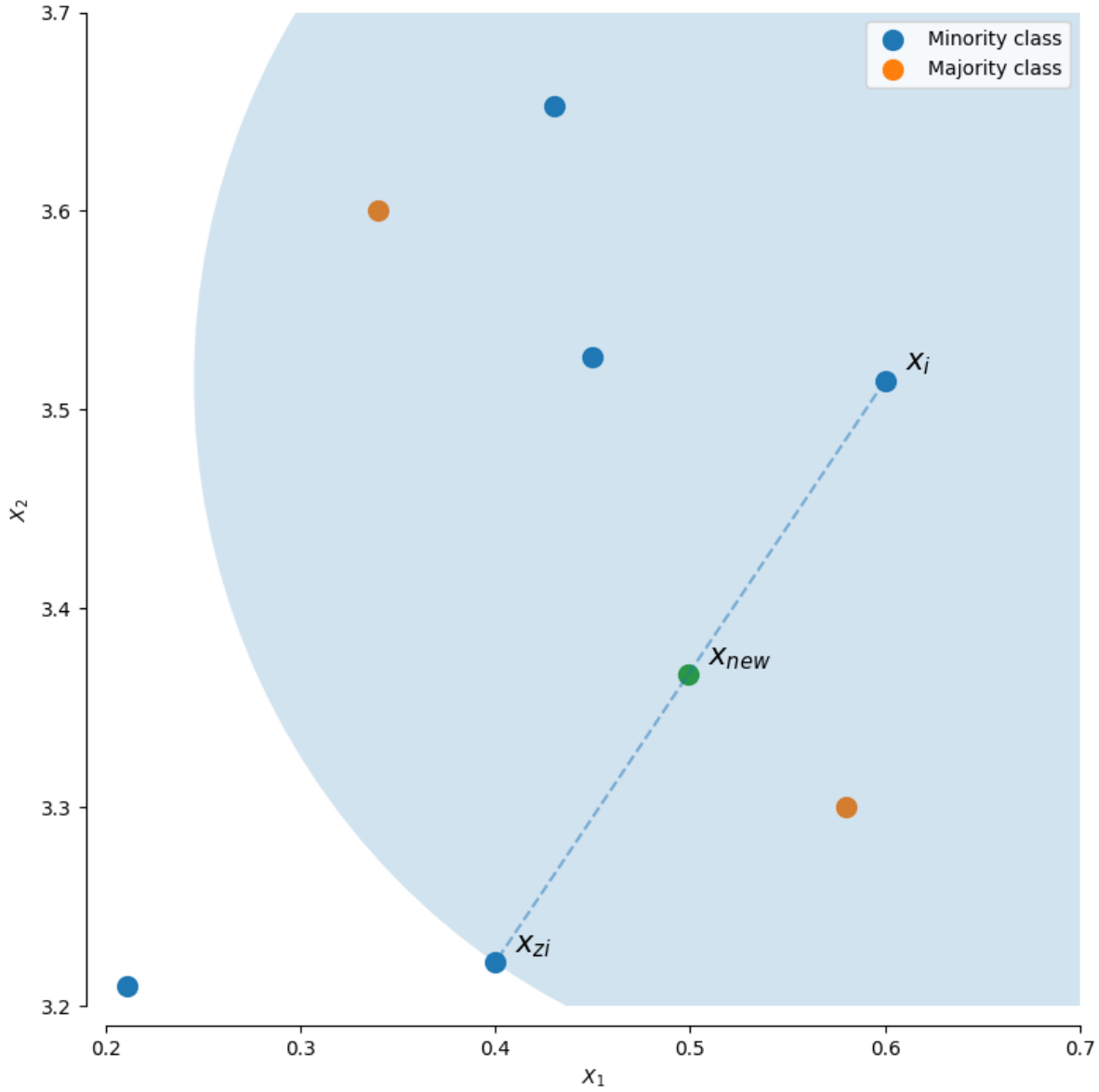
- Random Oversampling
  - Randomly sample the minority classes and simply duplicate the sampled observations



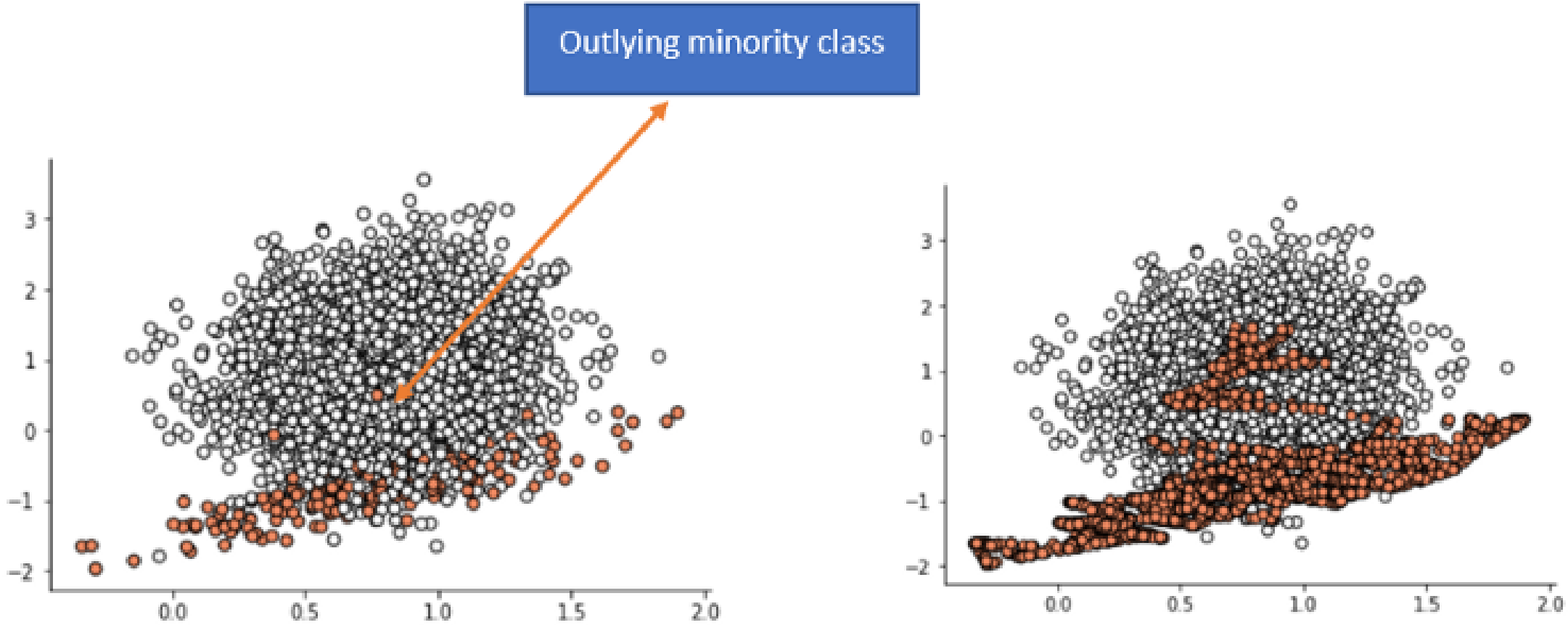
Synthetic Minority Over-sampling Technique (SMOTE)

- It generates new observations by interpolating between observations in the original dataset
- For a given observation x_i, a new (synthetic) observation is generated by interpolating between one of the k-nearest neighbors, x_{zi}.

$$x_{new} = x_i + \lambda(x_{zi} - x_i)$$



Problems with SMOTE



Introducing ADASYN

ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning

Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li

***Abstract*—This paper presents a novel adaptive synthetic (ADASYN) sampling approach for learning from imbalanced data sets. The essential idea of ADASYN is to use a weighted distribution for different minority class examples according to their level of difficulty in learning, where more synthetic data is generated for minority class examples that are harder to learn compared to those minority examples that are easier to learn. As a result, the ADASYN approach improves learning with respect to the data distributions in two ways: (1) reducing the bias introduced by the class imbalance, and (2) adaptively shifting the classification decision boundary toward the difficult examples. Simulation analyses on several machine learning data sets show the effectiveness of this method across five evaluation metrics.**

Generally speaking, imbalanced learning occurs whenever some types of data distribution significantly dominate the instance space compared to other data distributions. In this paper, we focus on the two-class classification problem for imbalanced data sets, a topic of major focus in recent research activities in the research community. Recently, theoretical analysis and practical applications for this problem have attracted a growing attention from both academia and industry. This is reflected by the establishment of several major workshops and special issue conferences, including the American Association for Artificial Intelligence workshop on Learning from Imbalanced Data Sets (AAAI'00) [9], the International Conference on Machine Learning workshop on Learning from

(1) Calculate the degree of class imbalance:

$$d = m_s/m_l \tag{1}$$

where $d \in (0,\ 1]$.
(2) If $d < d_{th}$ then (d_{th} is a preset threshold for the maximum tolerated degree of class imbalance ratio):

(a) Calculate the number of synthetic data examples that need to be generated for the minority class:

$$G = (m_l - m_s) \times \beta \tag{2}$$

Where $\beta \in [0,\ 1]$ is a parameter used to specify the desired balance level after generation of the synthetic data. $\beta = 1$ means a fully balanced data set is created after the generalization process.

(b) For each example $\boldsymbol{x_i} \in minorityclass$, find K nearest neighbors based on the Euclidean distance in n dimensional space, and calculate the ratio r_i defined as:

$$r_i = \Delta_i/K, \quad i = 1, ..., m_s \tag{3}$$

where Δ_i is the number of examples in the K nearest neighbors of $\boldsymbol{x}_i$ that belong to the majority class, therefore $r_i \in [0,\ 1]$;

(c) Normalize r_i according to $\hat{r}_i = r_i / \sum_{i=1}^{m_s} r_i$, so that $\hat{r}_i$ is a density distribution ($\sum_i \hat{r}_i = 1$)

(d) Calculate the number of synthetic data examples that need to be generated for each minority example $\boldsymbol{x}_i$:

$$g_i = \hat{r}_i \times G \tag{4}$$

where G is the total number of synthetic data examples that need to be generated for the minority class as defined in Equation (2).

(e) For each minority class data example $\boldsymbol{x}_i$, generate g_i synthetic data examples according to the following steps:

Do the **Loop** from 1 to g_i:

(i) Randomly choose one minority data example, $\boldsymbol{x}_{zi}$, from the K nearest neighbors for data $\boldsymbol{x}_i$.

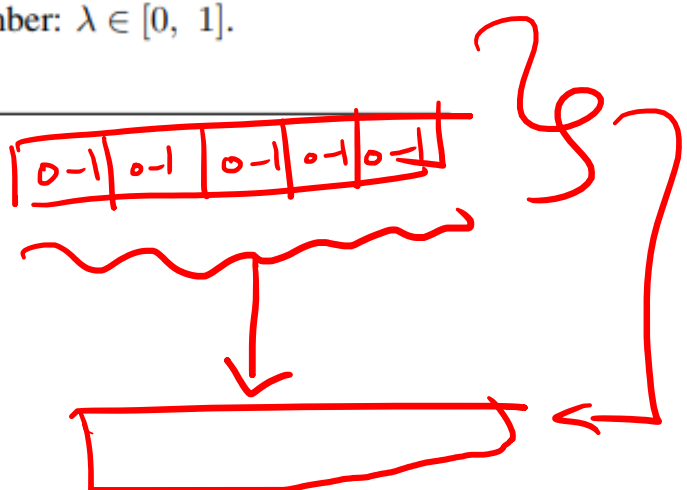
(ii) Generate the synthetic data example:

$$\boldsymbol{s_i} = \boldsymbol{x_i} + (\boldsymbol{x_{zi}} - \boldsymbol{x_i}) \times \lambda \tag{5}$$

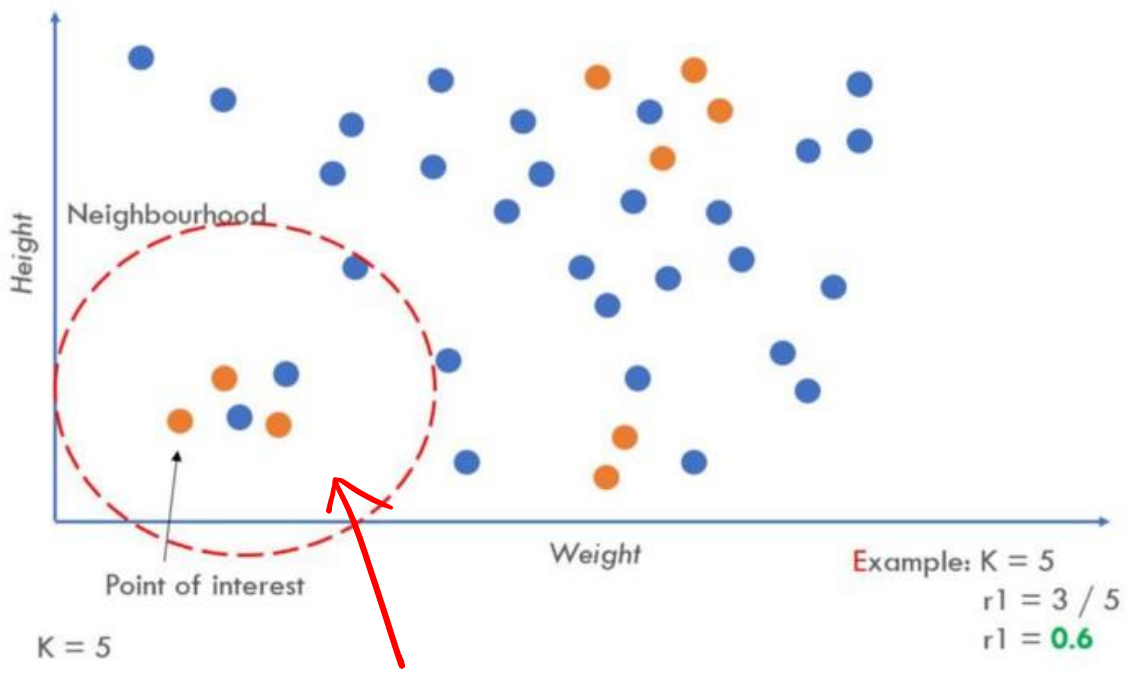
where $(\boldsymbol{x_{zi}} - \boldsymbol{x_i})$ is the difference vector in n dimensional spaces, and λ is a random number: $\lambda \in [0,\ 1]$.

End **Loop**

Procedure



Explanation



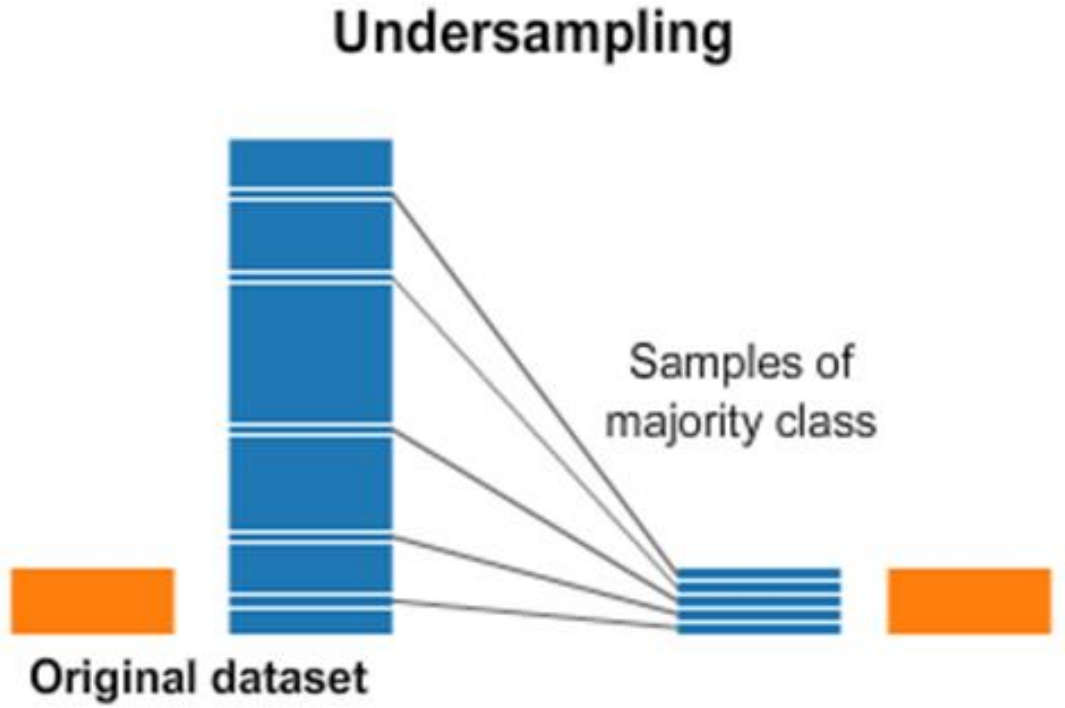
$2/5 = 0.4$

Minority Class	Minority Neighbours	Majority Neighbours	Impurity Ratio
Obs 1	3	2	.6
Obs 2	4	1	.4
Obs 3	1	4	.8
Obs 4	5	0	0

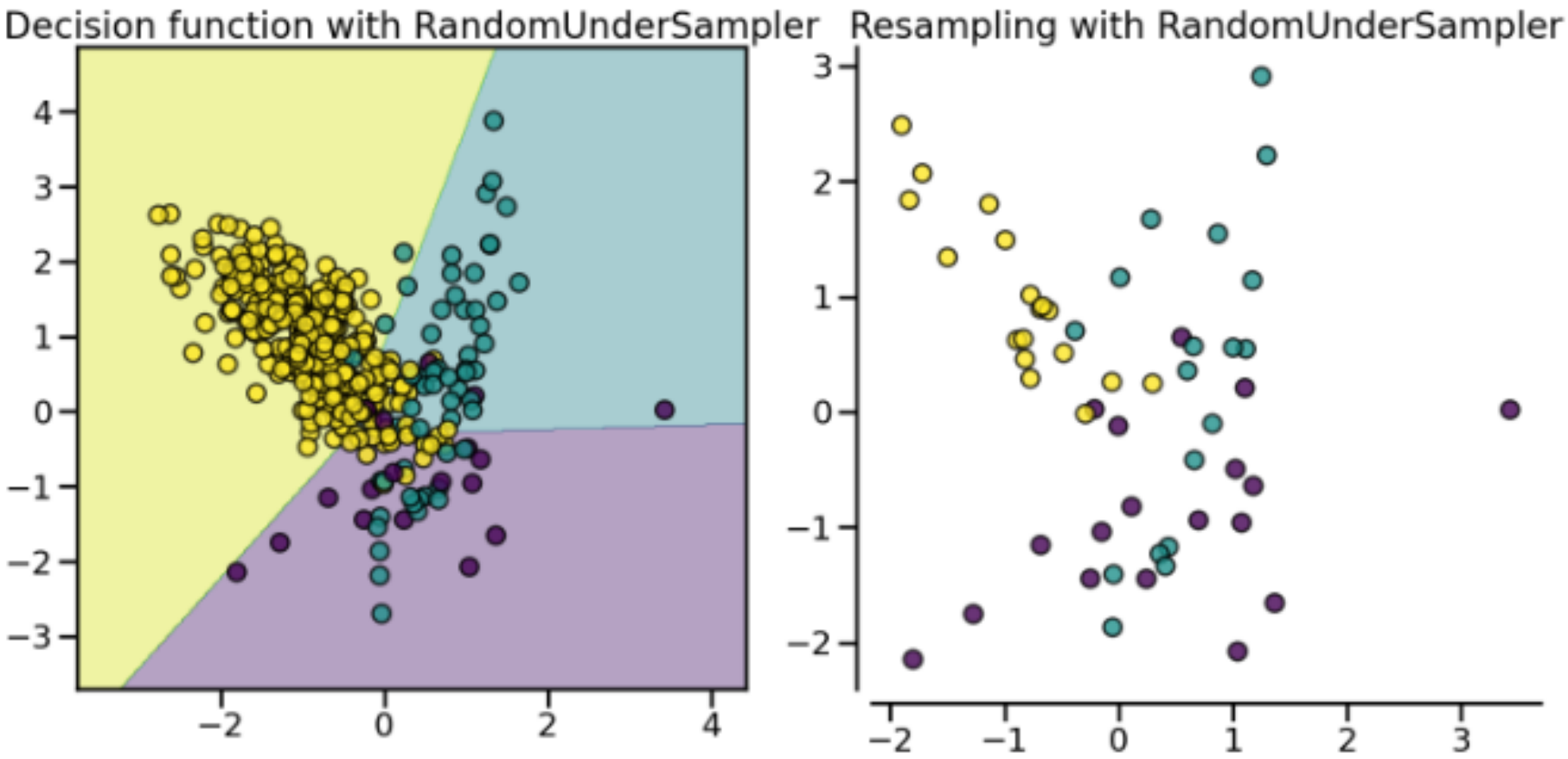
Under Sampling

- Throwing away data to make it easier to learn characteristics about the minority classes

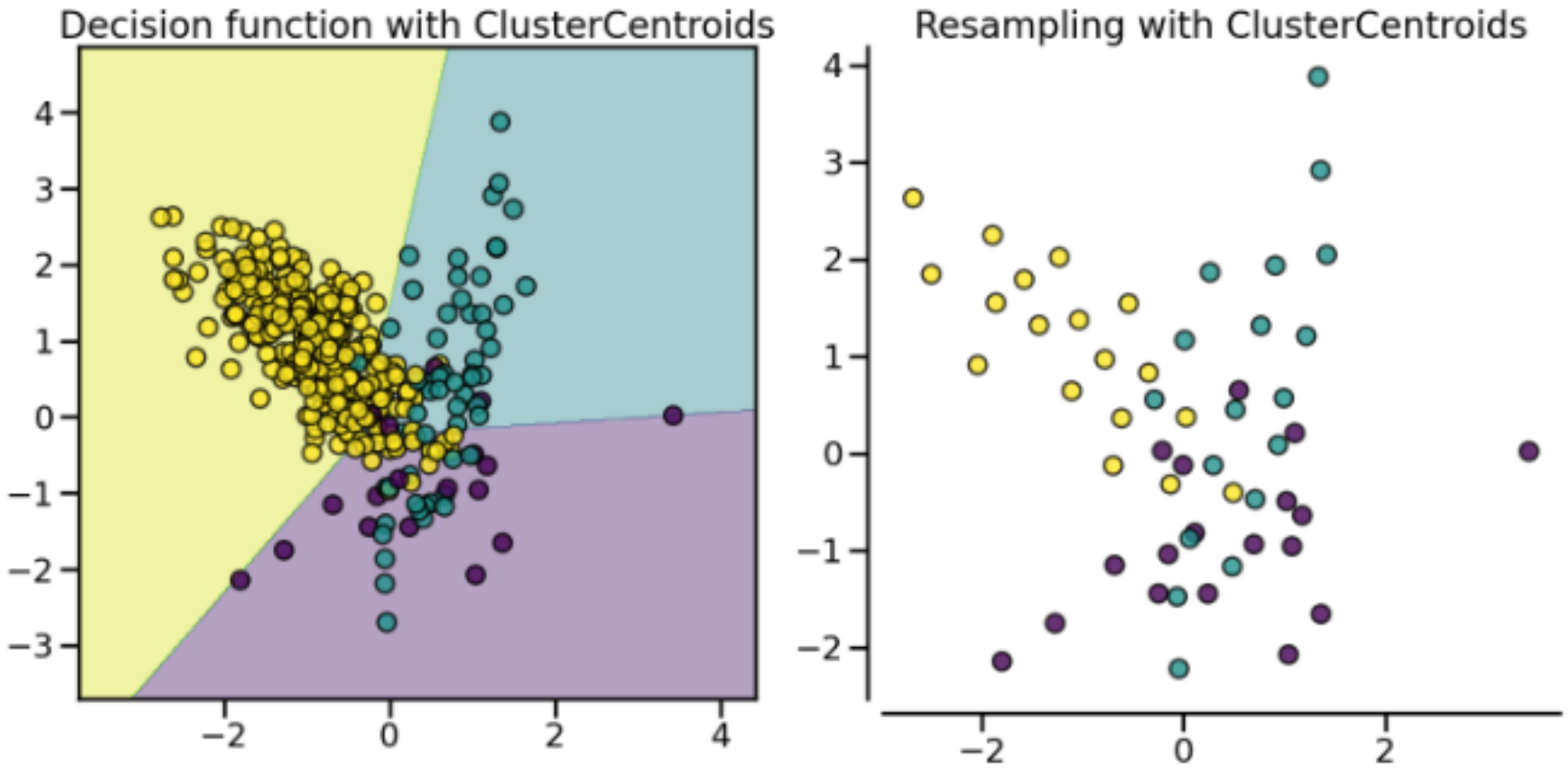
- **Random under sampling**
  - simply sample the majority class at random until reaching a similar number of observations as the minority classes



Random Undersampling



Replacement with cluster centroids



Tomek Links

necessarily subsume any other column of A_i since duplicate columns are removed in Step 3-b.3. However, at some point in the reduction of P' to I', P' is reduced to some $I'y_j^*$, $P' \subseteq I'y_j^*$ and, as shown above, there exists an FPI Q such that Q is the disjunction of $Q'\bar{y}_j^*$ and possibly X_i and/or $\bar{X}_i$. Q' may also have been reduced to Q'' by Step 3-b, $Q' \subseteq Q''$. So now,

$$A_i = X_i\bar{X}_i \begin{array}{c} \\ \\ \cdots \\ \\ \\ \end{array} \begin{array}{c} \text{col 1} \\ I' \\ \\ y_j^* \\ \\ \end{array} \begin{array}{c} \text{col 2} \\ Q'' \\ \\ \bar{y}_j^* \\ \\ \end{array} \begin{array}{c} \\ \\ \cdots \\ \\ \\ \end{array}$$

where col 1 is derived from P and col 2 is derived from Q. Since we are interested in generating $I'X_i\bar{X}_i$, we will consider two cases.

Case 1: $I' \subset Q''$. If P contains $X_i\bar{X}_i$, then an e is placed below col 1 and $X_i\bar{X}_iI' = I$ is added to A. However, $P = X_i\bar{X}_iP' \subseteq X_i\bar{X}_iy_j^*I' \subset X_i\bar{X}_iI' = I$. Therefore, $P \subset I$, which is a contradiction since P is an FPI. Therefore, P cannot contain $X_i\bar{X}_i$. So an $\tilde{e}$ is placed below col 1, and I is not added to A.

Case 2: $I' = Q''$. As shown above, P does not contain $X_i\bar{X}_i$.

Two Modifications of CNN

IVAN TOMEK

***Abstract*—The condensed nearest-neighbor (CNN) method chooses samples randomly. This results in a) retention of unnecessary samples and b) occasional retention of internal rather than boundary samples. Two modifications of CNN are presented which remove these disadvantages by considering only points close to the boundary. Performance is illustrated by an example.**

Introduction

The condensed nearest-neighbor (CNN) method [1] is a method of preprocessing of the design set for pattern recognition. It is based on the nearest-neighbor (NN) rule [2]. Its purpose is to reduce the size of the original design set D (set of samples with known membership) by the elimination of certain samples without affecting significantly the performance of NN classification: The NN rule used with E (the new design set $E \subset D$), should give almost the same result as NN used with D.

CNN works as follows:

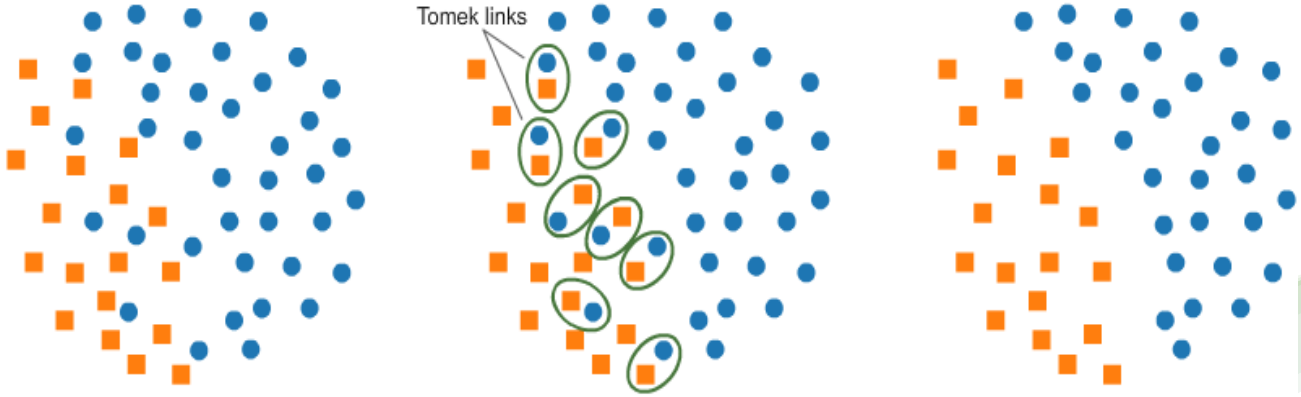
Ivan, Tomek. "Two modifications of CNN." *IEEE transactions on Systems, Man and Communications, SMC* 6 (1976): 769-772.

Tomek Link Removal

Let d(x_i, x_j) denotes the Euclidean distance between x_i and x_j, where x_i denotes sample that belongs to the minority class and x_j denotes sample that belongs to the majority class. If there is no sample x_k satisfies the following condition:

d(x_i, x_k) < d(x_i, x_j), or
d(x_j, x_k) < d(x_i, x_j)

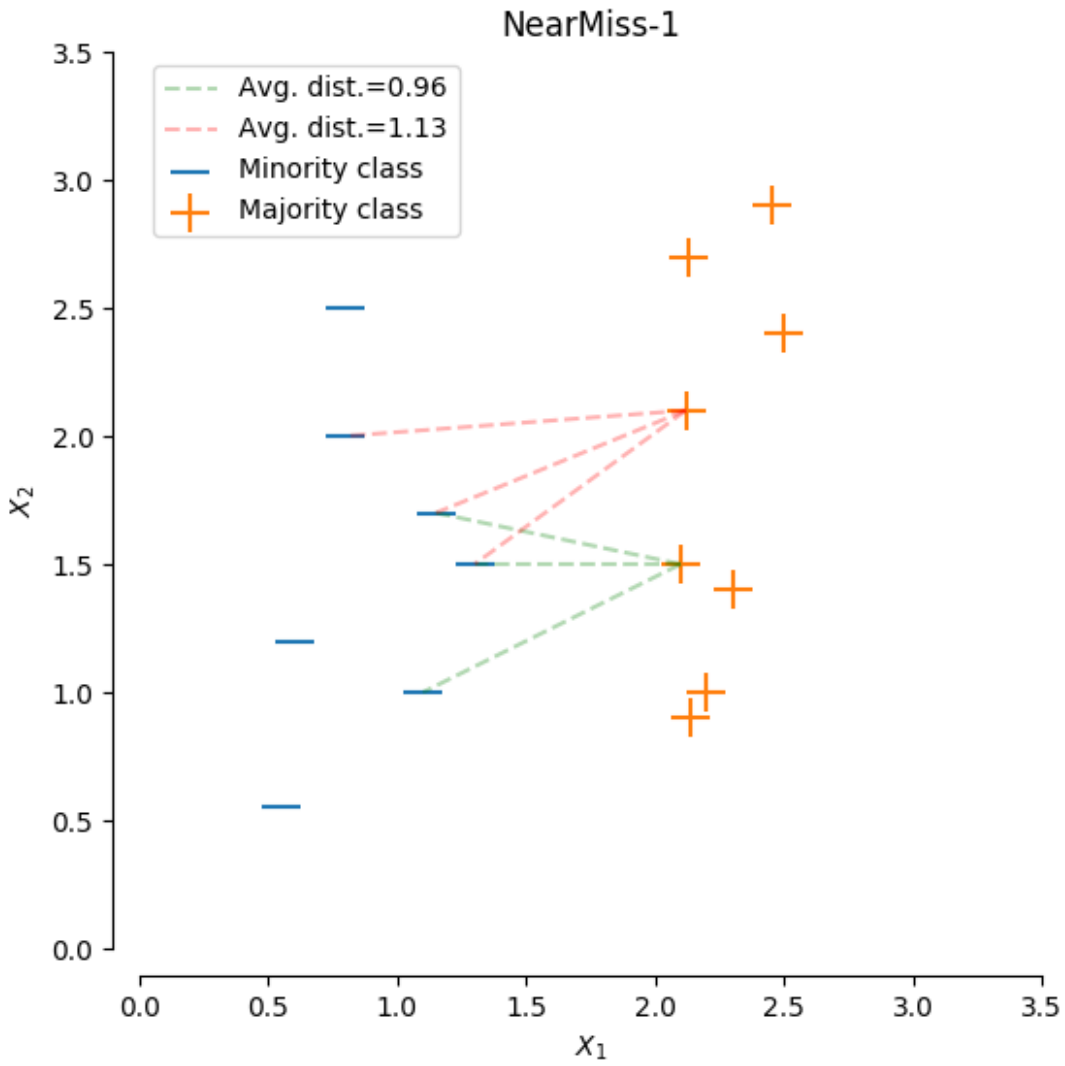
then the pair of (x_i, x_j) is a ***Tomek Link****.*



Source: https://hersanyagci.medium.com/under-sampling-methods-for-imbalanced-data-clustercentroids-randomundersampler-nearmiss-eae0eadcc145

Near miss -1

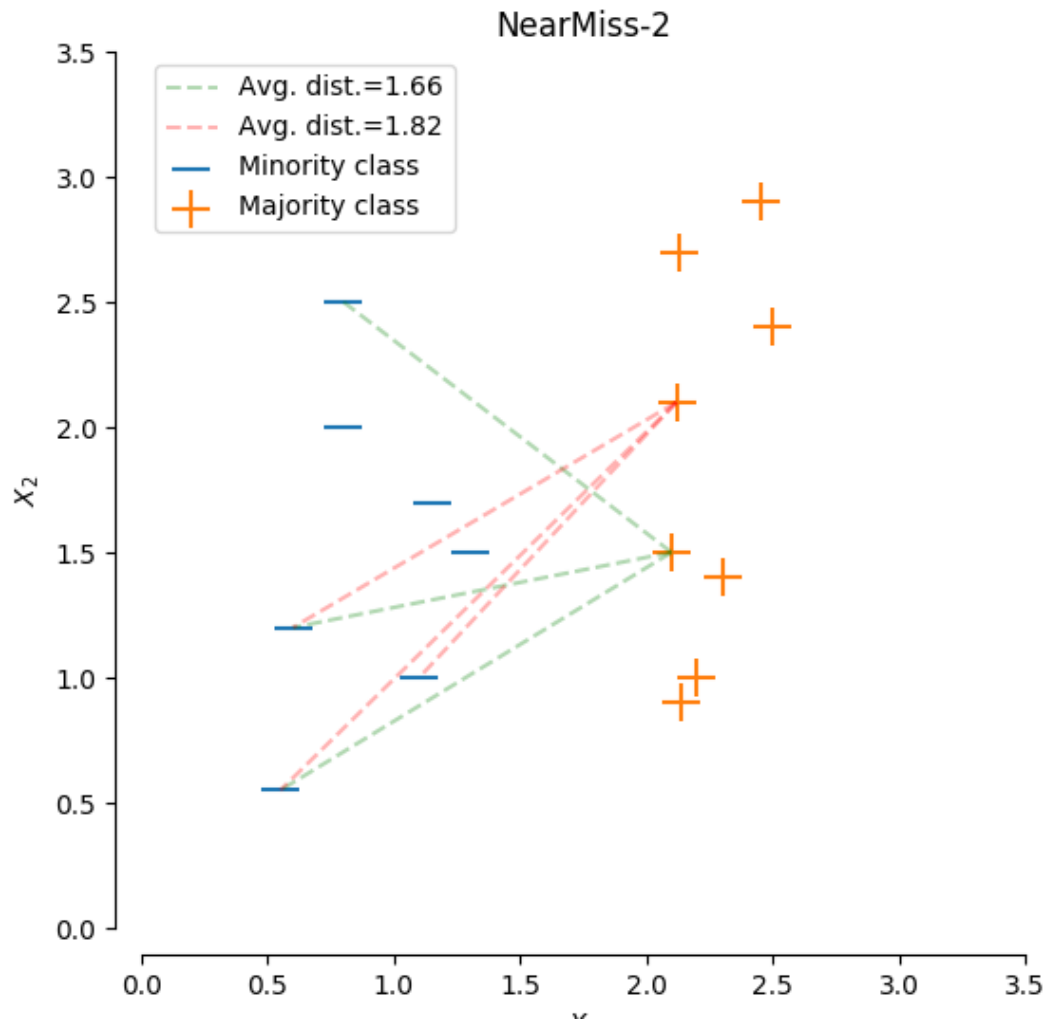
NearMiss-1 select samples from the majority class for which the average distance of the N closest samples of a minority class is smallest.



Source: https://hersanyagci.medium.com/under-sampling-methods-for-imbalanced-data-clustercentroids-randomundersampler-nearmiss-eae0eadcc145

Near miss -2

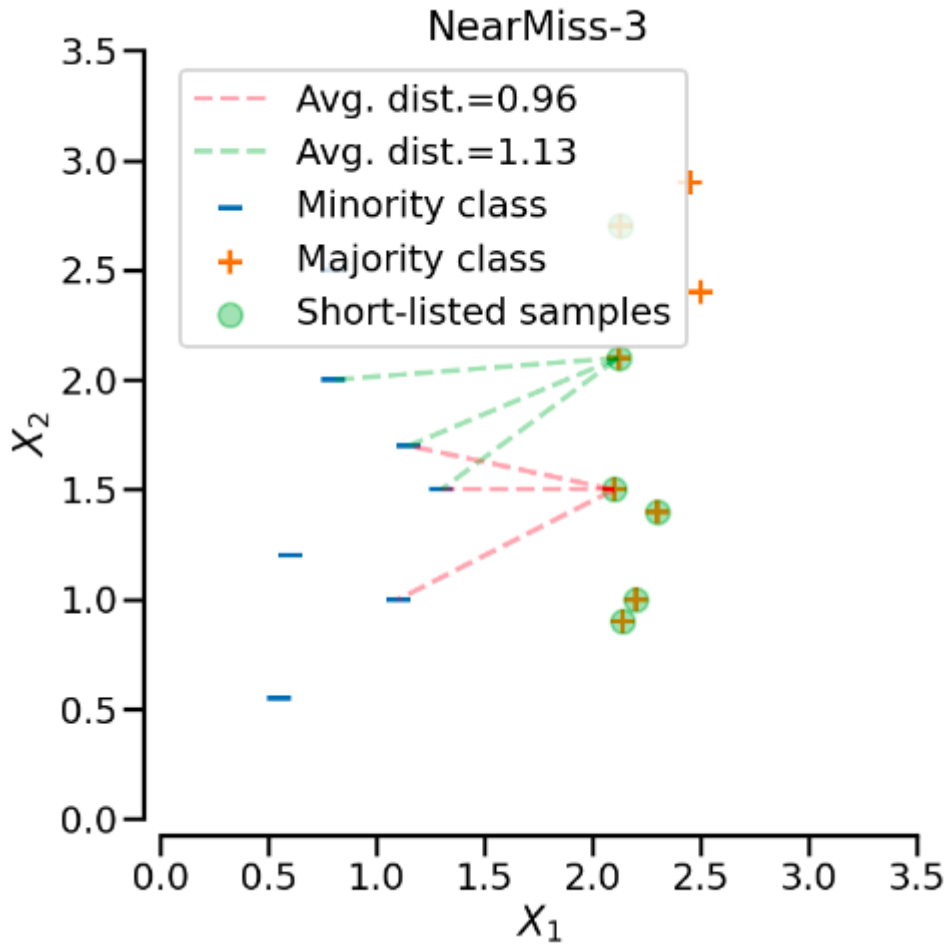
Select samples from the majority class for which the average distance of the N farthest samples of a minority class is smallest.



Source: https://hersanyagci.medium.com/under-sampling-methods-for-imbalanced-data-clustercentroids-randomundersampler-nearmiss-eae0eadcc145

Near miss -3

NearMiss-3 is a 2-steps algorithm. First, for each negative sample, their m nearest-neighbors will be kept. Then, the positive samples selected are the ones for which the average distance to the k nearest-neighbors is the largest..



Performance Comparison

Table 2. The performances (F-measure, G-mean, AUC, and accuracy) of 8 classifiers for diabetes. The evaluation metrics obtained by preprocessing with SMOTE combined with Tomek links, with only SMOTE and without any preprocessing procedure are denoted by cases 1, 2, and 3, respectively.

	F-Measure			G-mean			AUC			Acc (%)		
	1	2	3	1	2	3	1	2	3	1	2	3
SVM	**0.773**	0.753	0.763	**0.775**	0.753	0.762	**0.768**	0.753	0.720	**77.459**	75.290	77.344
BN	**0.783**	0.768	0.742	**0.780**	0.771	0.713	**0.864**	0.851	0.806	**78.232**	76.931	74.349
AdaBoost	**0.775**	0.754	0.738	**0.775**	0.754	0.716	**0.848**	0.821	0.801	**77.569**	75.386	74.349
K*	**0.804**	0.802	0.683	0.801	**0.806**	0.650	**0.886**	0.873	0.714	**80.332**	80.309	69.141
C4.5	**0.775**	0.758	0.736	**0.772**	0.759	0.707	**0.813**	0.793	0.751	**77.459**	75.869	73.828
RBF Network	**0.755**	0.726	0.746	**0.754**	0.726	0.732	**0.832**	0.804	0.783	**75.580**	72.587	75.391
LR	**0.781**	0.754	0.765	**0.782**	0.754	0.729	**0.864**	0.838	0.832	**78.232**	75.386	77.214
LMT	**0.792**	0.752	0.766	**0.789**	0.753	0.760	**0.859**	0.831	0.831	**79.116**	75.290	77.474

Zeng, Min, et al. "Effective prediction of three common diseases by combining SMOTE with Tomek links technique for imbalanced medical data." *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*. IEEE, 2016.

Performance Continued

Table 3. The performances (F-measure, G-mean, AUC, and accuracy) of 8 classifiers with 3 cases (the same as Table 3) for Parkinson's disease

	F-Measure			G-mean			AUC			Acc (%)		
	1	2	3	1	2	3	1	2	3	1	2	3
SVM	0.851	0.841	**0.856**	0.865	0.849	**0.908**	**0.849**	0.843	0.747	85.252	84.192	**87.180**
BN	**0.833**	0.814	0.810	**0.840**	0.817	0.723	**0.935**	0.918	0.871	**83.453**	81.443	80.000
AdaBoost	**0.910**	0.828	0.846	**0.911**	0.827	0.808	**0.952**	0.917	0.889	**91.007**	82.818	85.128
K*	**0.935**	0.931	0.900	0.935	**0.937**	0.847	**0.991**	0.987	0.968	**93.525**	93.127	89.744
C4.5	**0.884**	0.880	0.804	**0.888**	0.880	0.726	**0.889**	0.883	0.769	**88.489**	87.973	80.513
RBF Network	0.829	**0.834**	0.830	**0.841**	**0.841**	0.806	**0.883**	0.878	0.862	83.094	83.505	**84.103**
LR	0.831	0.828	**0.866**	**0.831**	0.828	0.818	0.924	**0.927**	0.883	83.094	82.818	**86.667**
LMT	**0.899**	0.893	0.856	**0.900**	0.894	0.825	**0.962**	0.951	0.835	**89.928**	89.347	86.15

Table 4. The performances (F-measure, G-mean, AUC, and accuracy) of 8 classifiers with 3 cases (the same as Table 3) for vertebral column

	F-Measure			G-mean			AUC			Acc (%)		
	1	2	3	1	2	3	1	2	3	1	2	3
SVM	**0.812**	0.811	0.770	**0.822**	0.818	0.786	**0.815**	0.814	0.704	**81.390**	81.220	78.710
BN	**0.825**	0.809	0.772	**0.833**	0.818	0.734	**0.874**	0.861	0.853	**82.630**	80.976	76.452
AdaBoost	**0.835**	0.830	0.803	**0.846**	0.842	0.765	0.912	**0.917**	0.894	**83.623**	83.171	80.000
K*	**0.867**	0.854	0.835	**0.883**	0.873	0.800	**0.976**	0.960	0.896	**86.849**	85.610	83.226
C4.5	**0.871**	0.824	0.812	**0.871**	0.824	0.795	**0.890**	0.858	0.838	**87.097**	82.439	81.613
RBF Network	**0.853**	0.846	0.805	**0.856**	0.849	0.769	**0.904**	0.895	0.871	**85.360**	84.634	80.323
LR	**0.861**	0.859	0.855	**0.862**	0.859	0.830	**0.944**	0.941	0.934	**86.104**	85.854	85.484
LMT	**0.866**	0.851	0.856	**0.868**	0.853	0.828	0.926	**0.935**	0.929	**86.601**	85.122	85.484

Image Data Augmentation

- Process of artificially increasing the amount of data by generating new data points from existing data.
- Done by adding minor alterations to data to generate new data points in the latent space of original data to amplify the dataset.

- **Synthetic data**: When data is generated artificially without using real-world images. Often produced by GANs.
- **Augmented data**: Derived from original images with some sort of minor geometric transformations (such as flipping, translation, rotation, or the addition of noise) in order to increase the diversity of the training set.

Image Data Augmentation

- In real-world cases, we might have a dataset of photos captured under a specific set of conditions.
- While our test data may exist in a number of variations, such as varied orientations, locations, scales, brightness, and so on. We can accommodate such cases by training deep neural networks with synthetically manipulated data.

- Popular in computer vision

Image Data Augmentation

- **Advantages:**
  - Brings diversity in data
  - Deals with the limited data problem
  - Improve model prediction
  - Reduce the cost of collecting and labeling data
  - Help resolve class imbalance problem
  - Increase generalization of the model
  - Reduced data overfitting

Image Data Augmentation

Image processing activities for data augmentation

- Padding
- Rotation
- Scaling
- Flipping (vertical and horizontal)
- Translation (image is moved along X, Y direction)
- Cropping
- Darkening & Brightening
- Gray scaling
- Changing contrast
- Adding noise

Image Data Augmentation

FLIP

Original *Horizontally flipped* *Vertically flipped*

Image Data Augmentation

Rotate

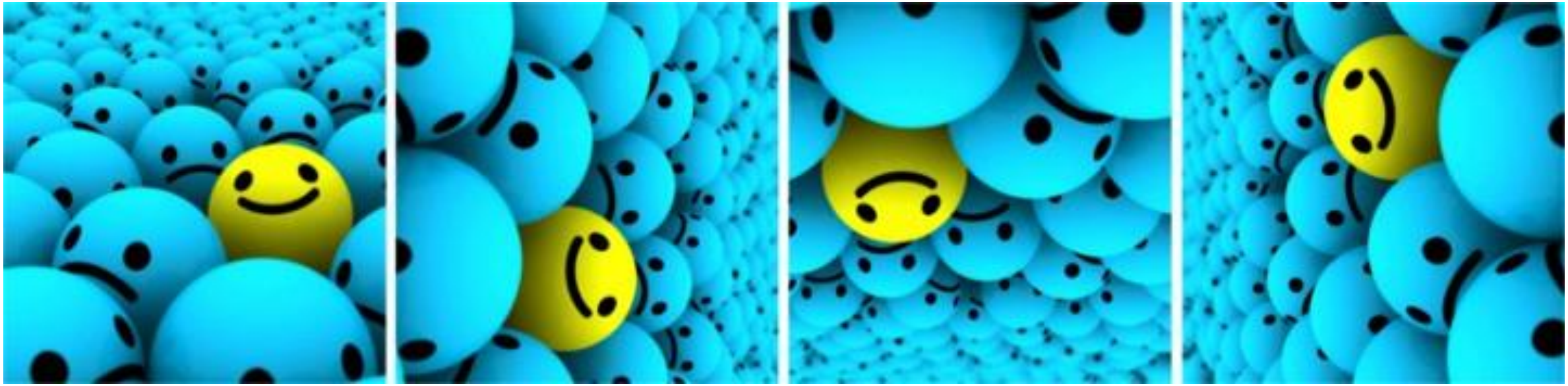


Image Data Augmentation

Scale

Original *Scaled out by 10%* *Scaled out by 20%*

Image Data Augmentation

Crop

Image Data Augmentation

Adding noise

Original *Gaussian noise* *Salt pepper noise*

Image Data Augmentation

Translation

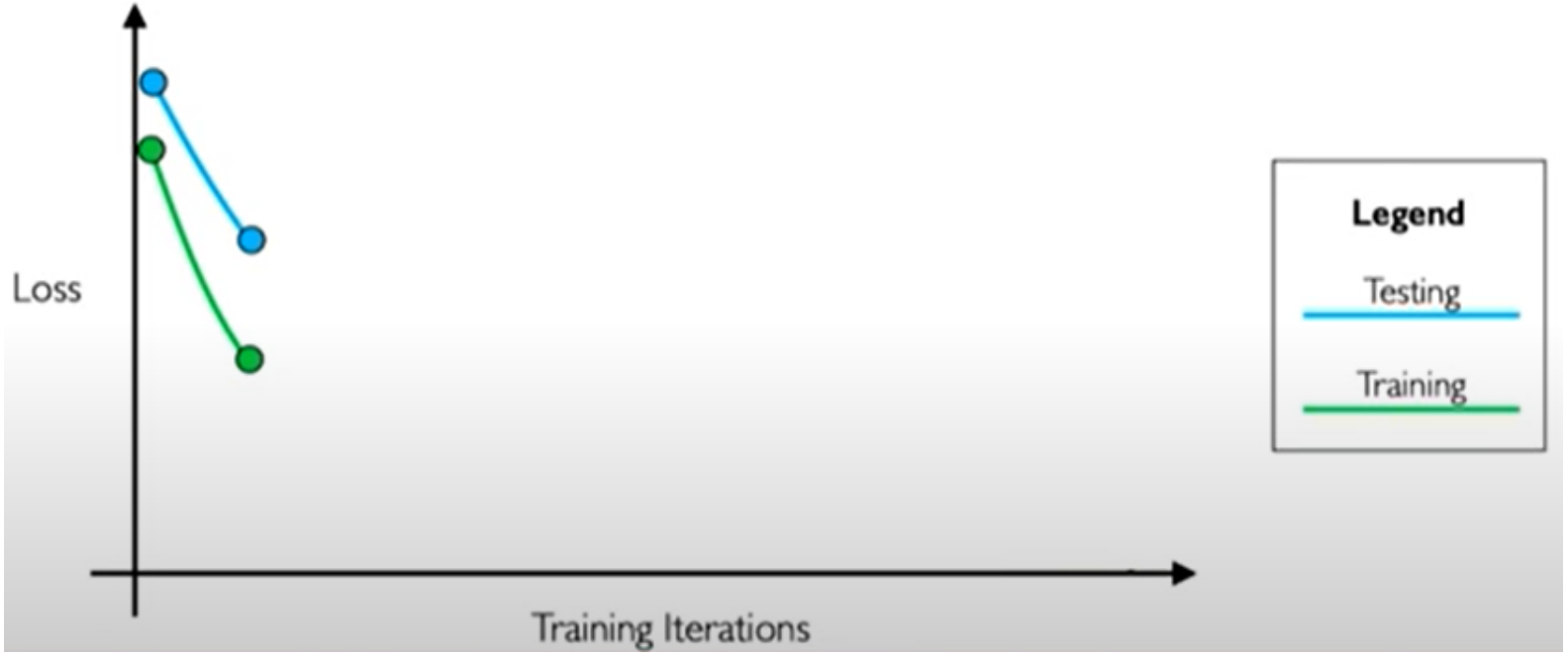
Original

Translated to right

Translated upwards

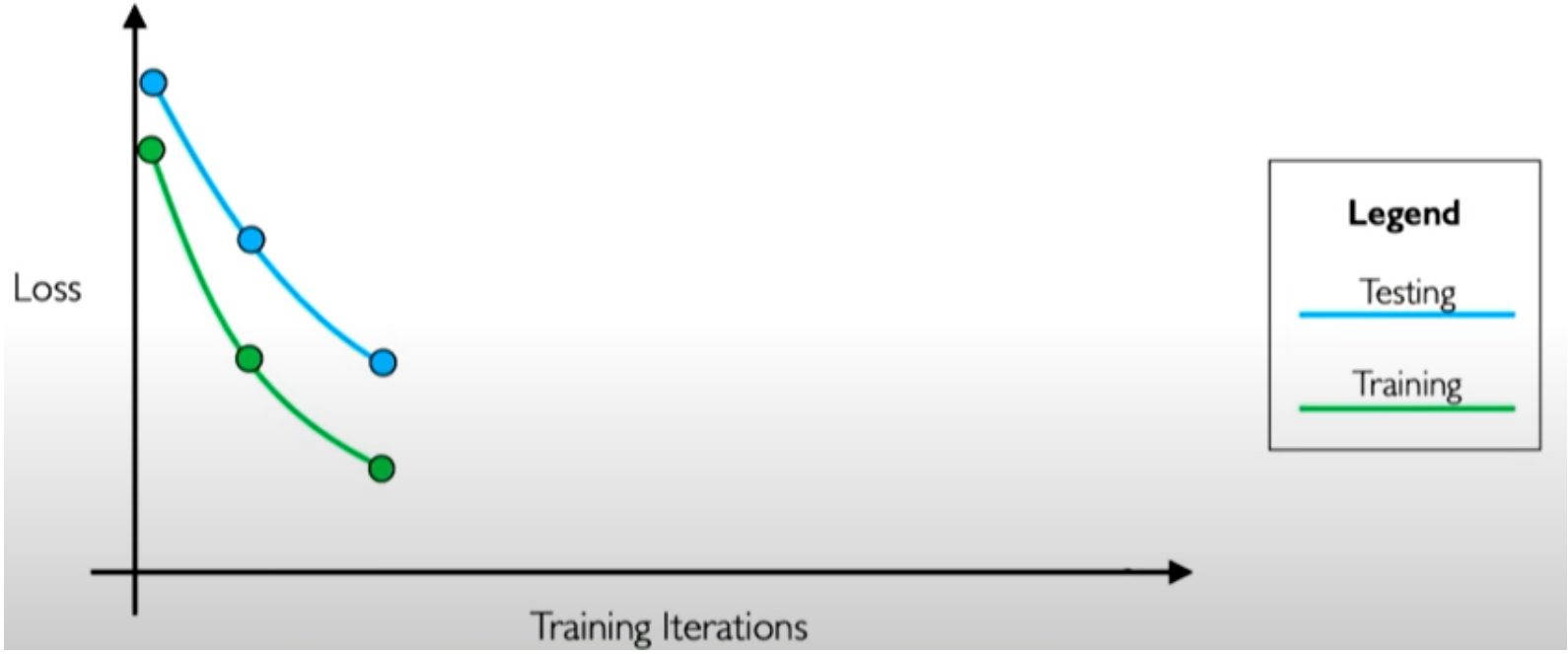
Regularization: Early Stopping

- Overfitting also happens when there is too much training.
- **Solution**: Stop training before we have a chance to overfit.



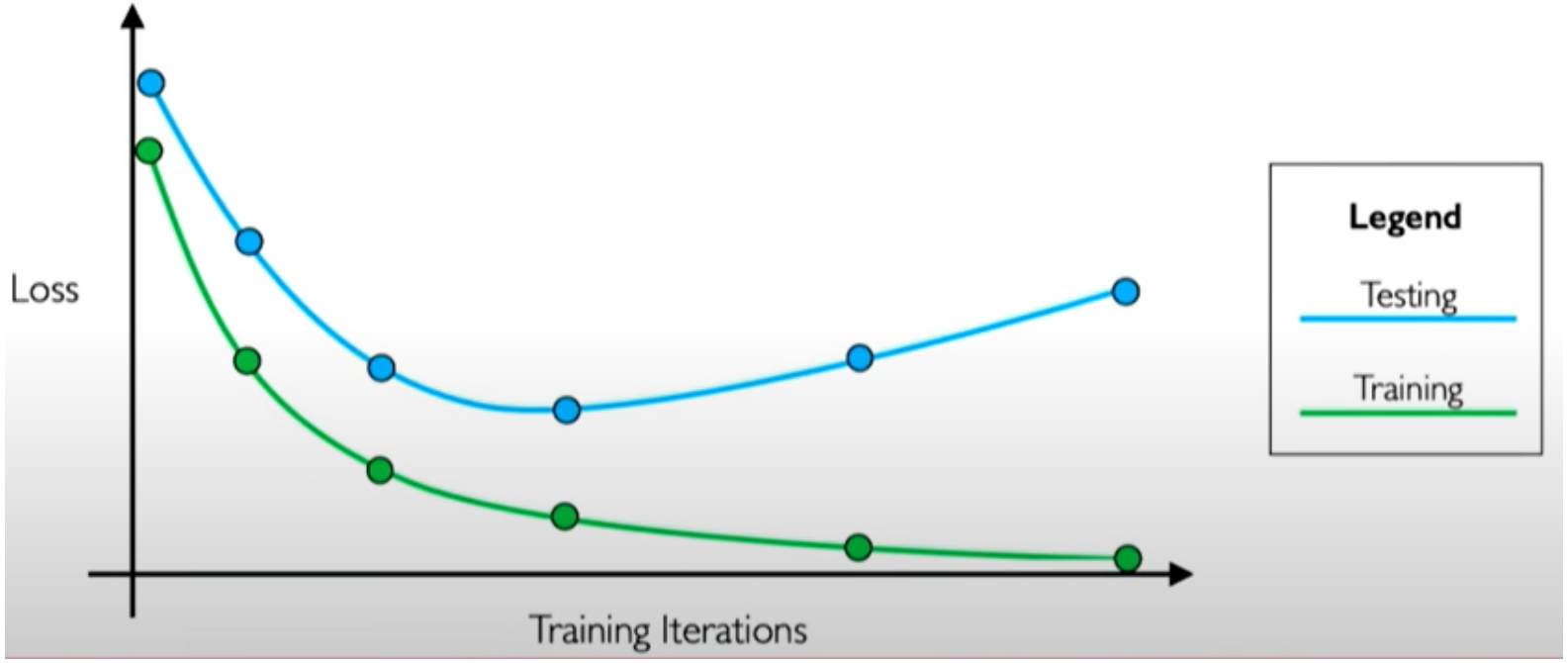
Regularization: Early Stopping

- Stop training before we have a chance to overfit



Regularization: Early Stopping

- Stop training before we have a chance to overfit



Regularization: Early Stopping

- Stop training before we have a chance to overfit

Normalization

- Recall the loss function

$$\mathcal{L}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the prediction for the sample $\mathbf{x}_i$:

$$\hat{y}_i = \mathbf{w}'\mathbf{x}_i$$

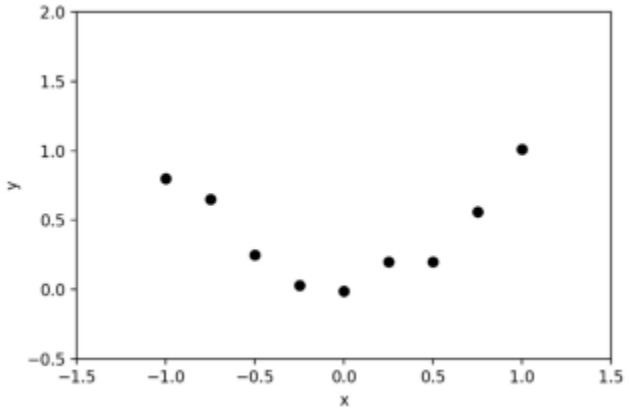
$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

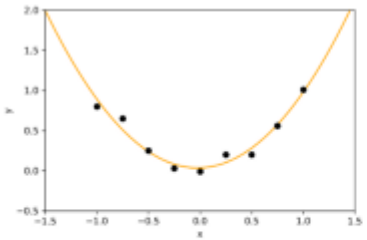
Some of the common Norms

$$\text{L1 norm: } \|\mathbf{w}\|_1 = \sum_i^n |w_i|$$

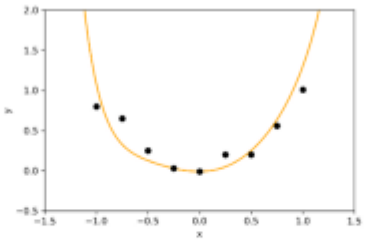
$$\text{Squared L2 norm: } \|\mathbf{w}\|_2^2 = \sum_i^n w_i^2$$

Curve Fitting

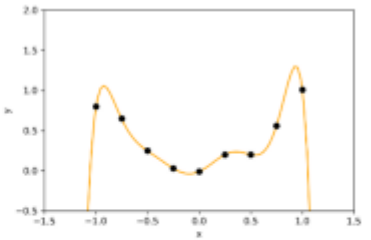




(a) #params = 3
MSE = 0.006
L2 norm = 0.90
L1 norm = 0.98

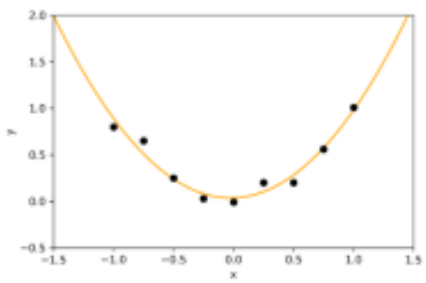


(b) #params = 9
MSE = 0.035
L2 norm = 1.06
L1 norm = 2.32

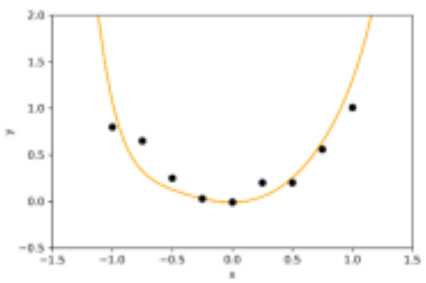


(c) #params = 9
MSE = 0
L2 norm = 32.69
L1 norm = 70.03

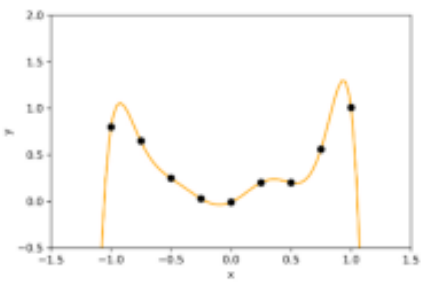
Figure a: $\hat{y} = 0.04 + 0.04x + 0.9x^2$
Figure b: $\hat{y} = -0.01 + 0.01x + 0.8x^2 + 0.5x^3 - 0.1x^4 - 0.1x^5 + 0.3x^6 - 0.3x^7 + 0.2x^8$
Figure c: $\hat{y} = -0.01 + 0.57x + 2.67x^2 - 4.08x^3 - 12.25x^4 + 7.41x^5 + 24.87x^6 - 3.79x^7 - 14.38x^8$



(a) 2nd-degree polynomial



(b) 8th-degree polynomial

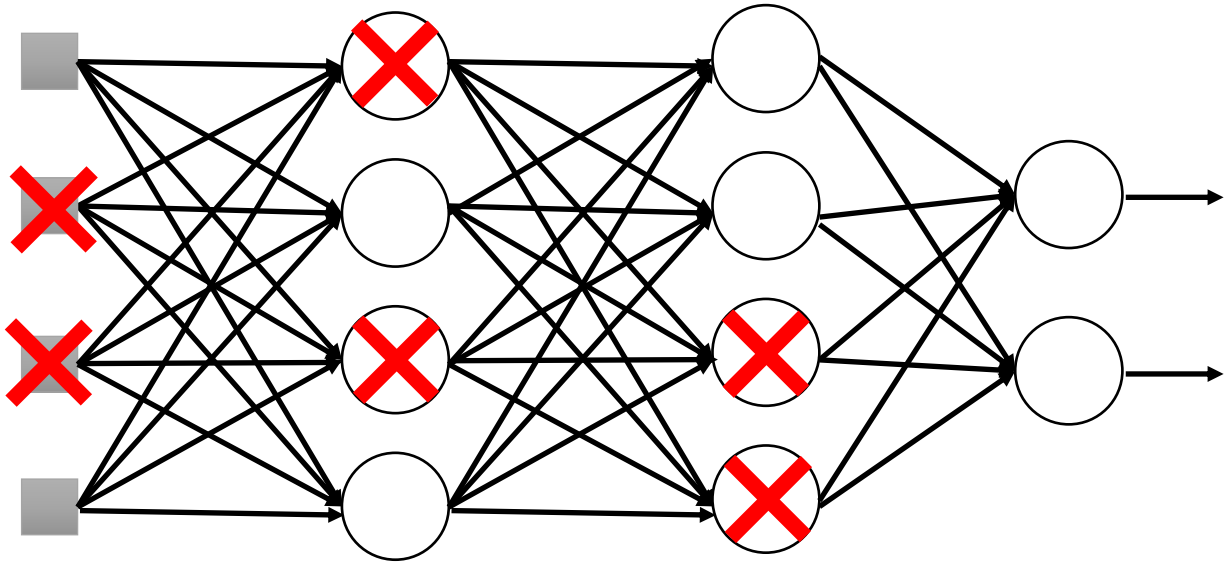


(c) Another 8th-degree polynomial

Source: Medium

Dropout

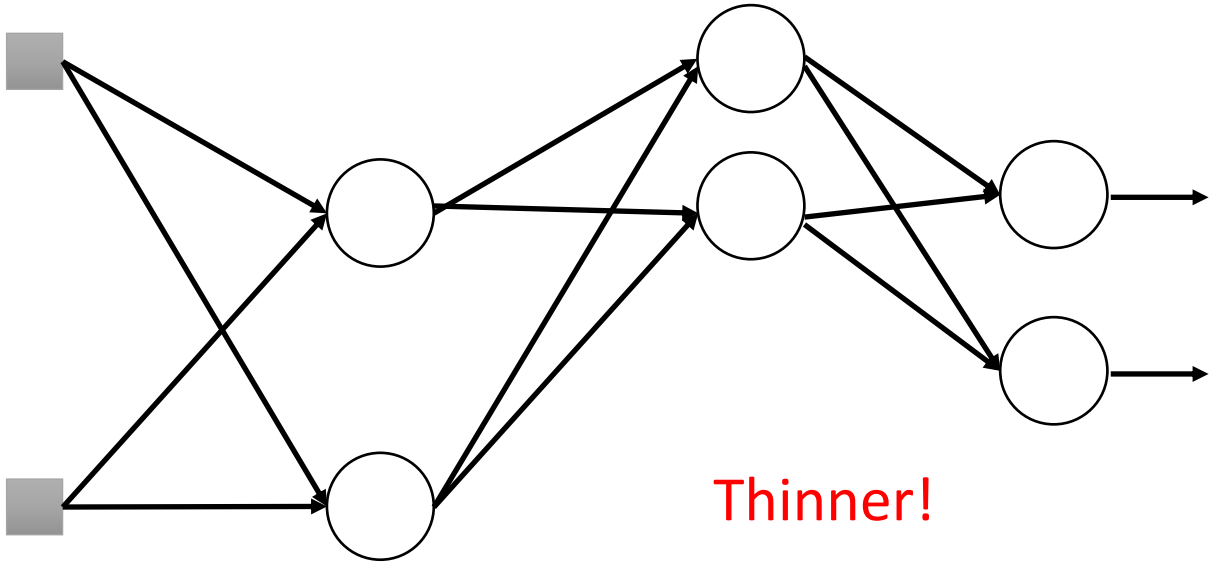
Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

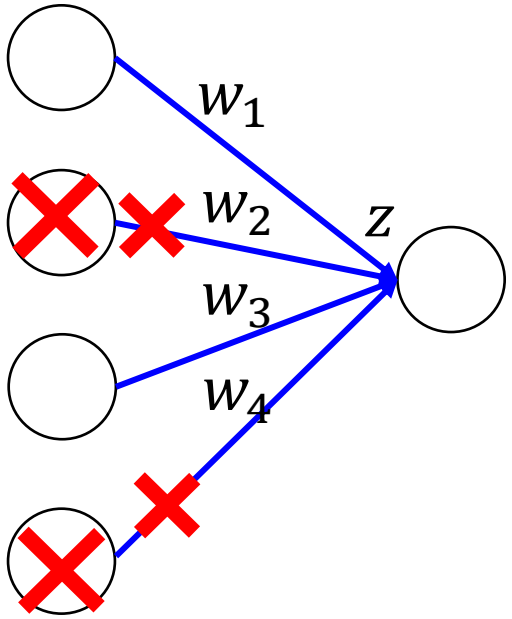
    The structure of the network is changed.
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

Dropout

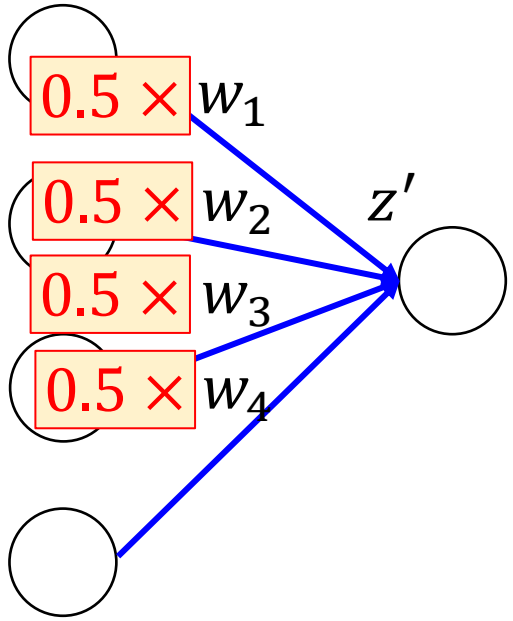
Training of Dropout

Assume dropout rate is 50%

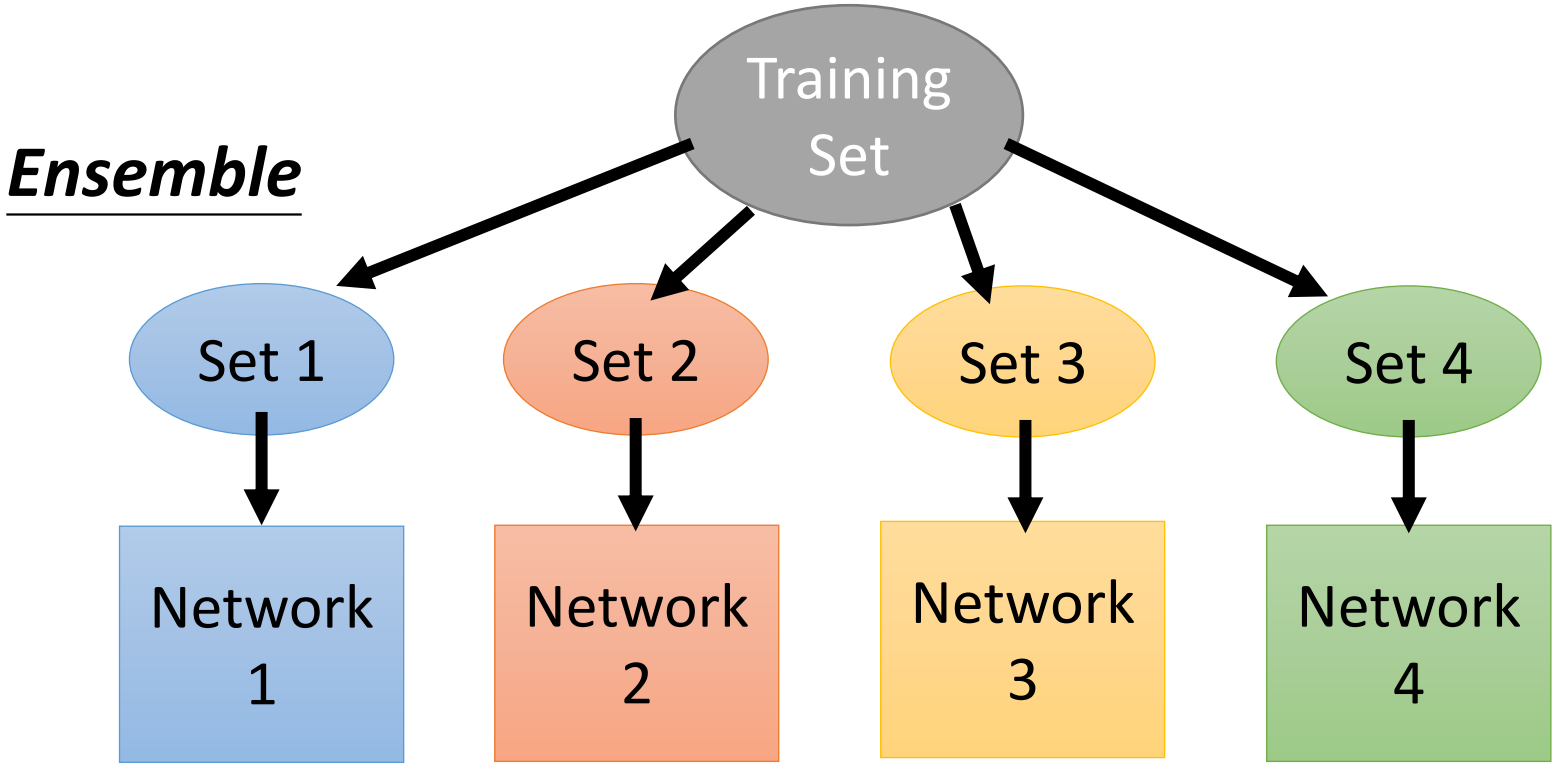


Testing of Dropout

No dropout



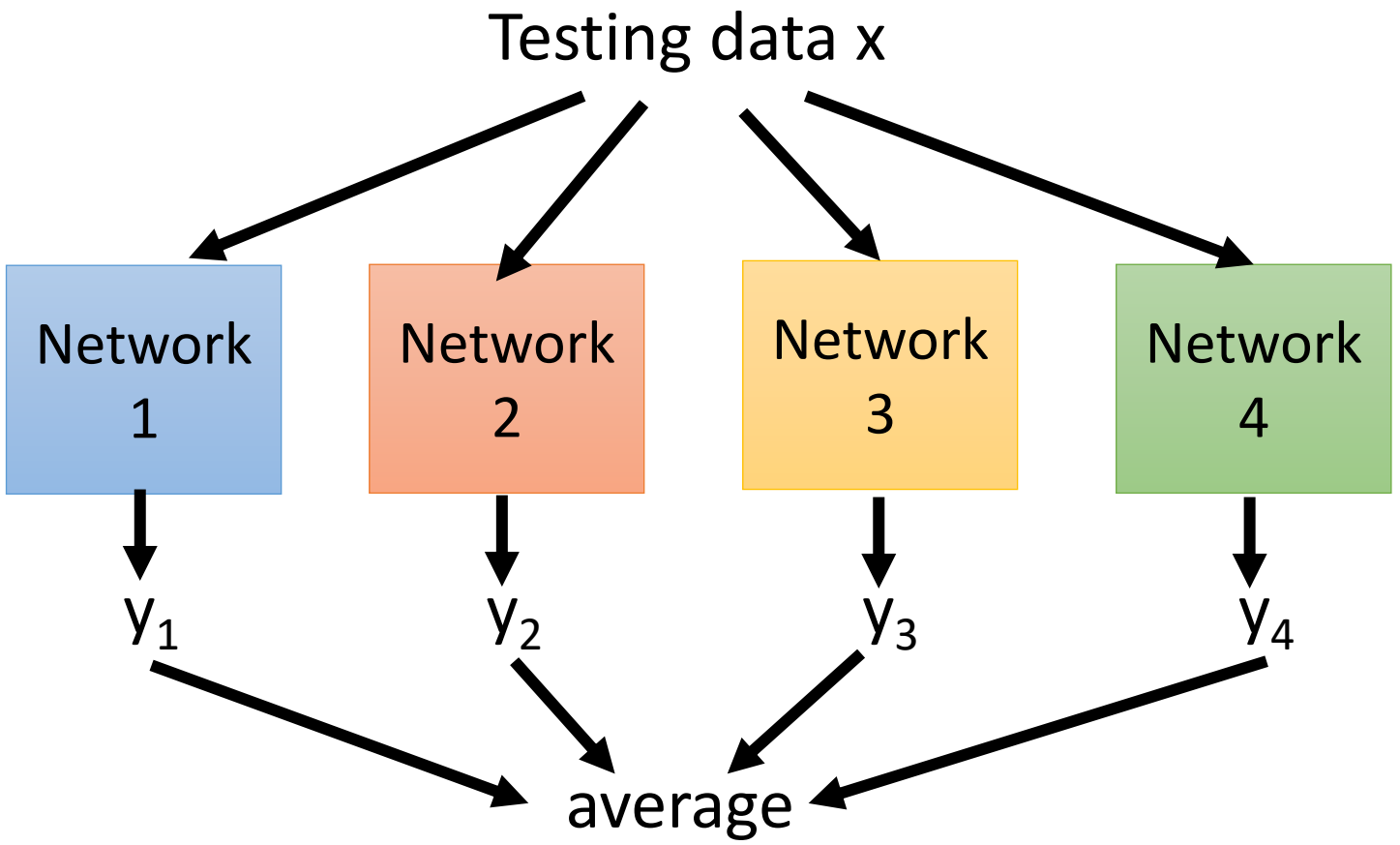
Dropout is a kind of ensemble.



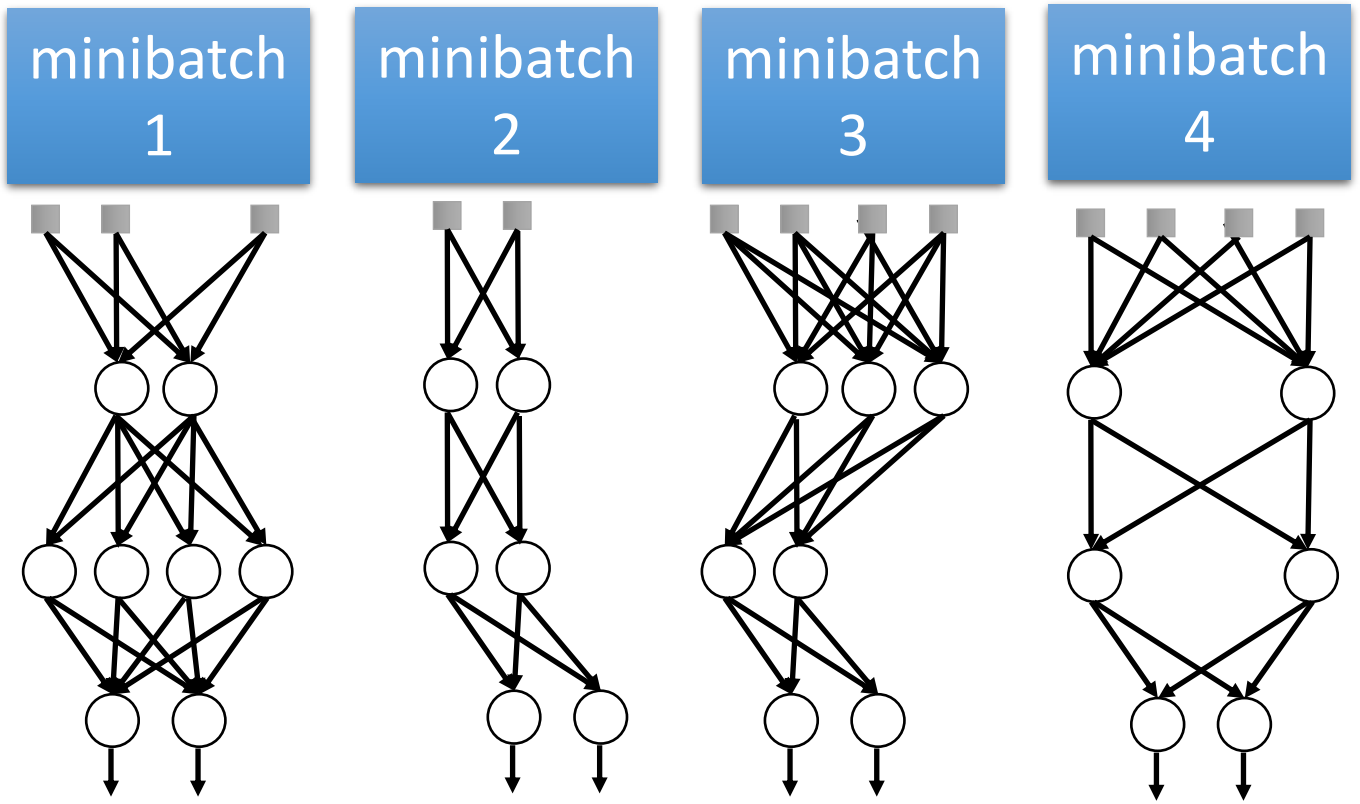
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



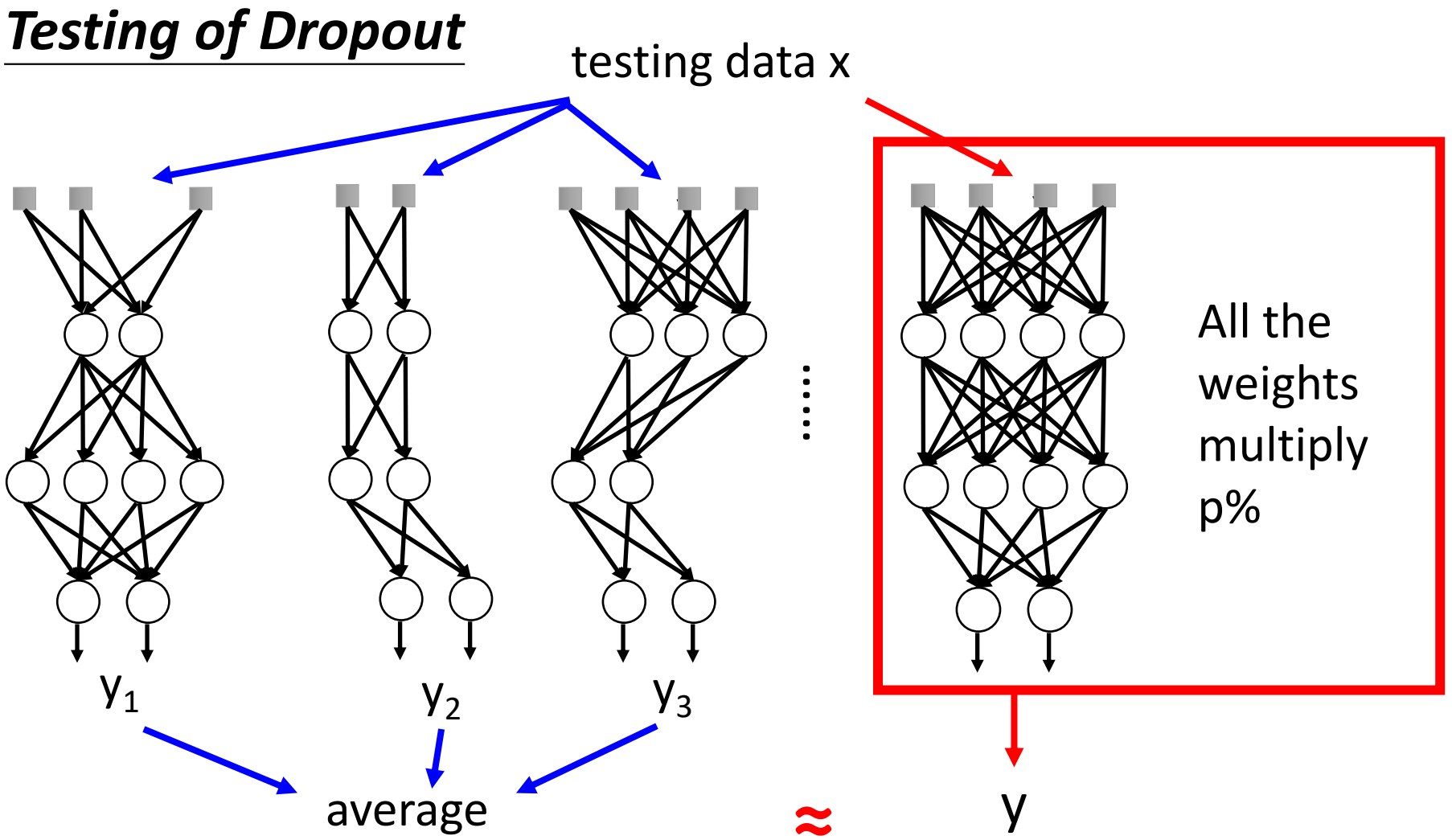
Training of Dropout

M neurons

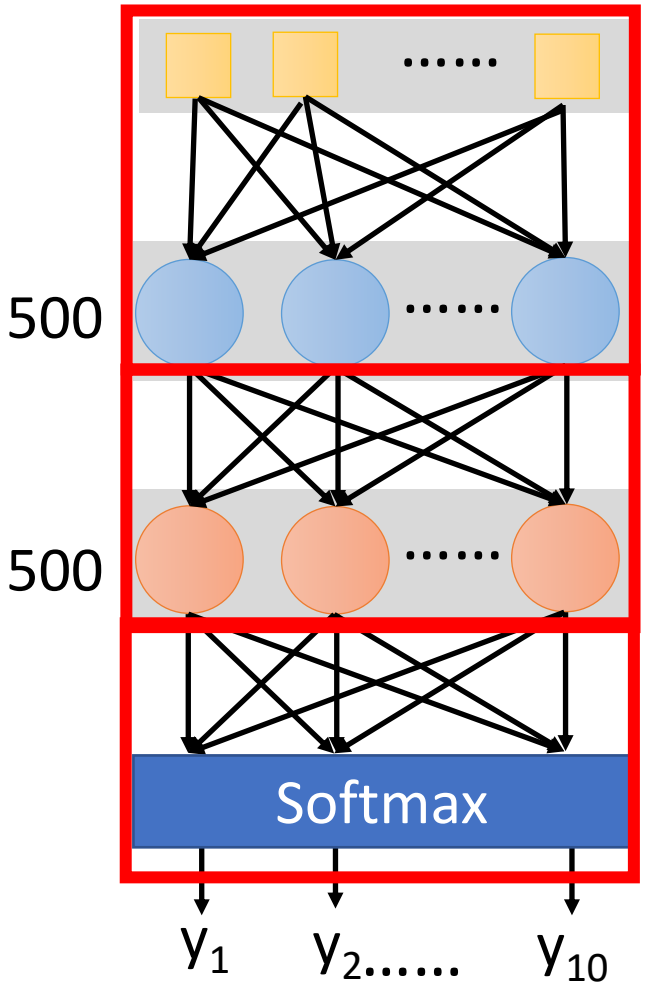
2^M possible networks

- Using one mini-batch to train one network
- Some parameters in the network are shared

Dropout is a kind of ensemble.



Demo




```
model = Sequential()
```

```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

Some Results

sets.

In order to test the robustness of dropout, classification experiments were done with networks of many different architectures keeping all hyperparameters, including p, fixed. Figure 4 shows the test error rates obtained for these different architectures as training progresses. The same architectures trained with and without dropout have drastically different test errors as seen as by the two separate clusters of trajectories. Dropout gives a huge improvement across all architectures, without using hyperparameters that were tuned specifically for each architecture.

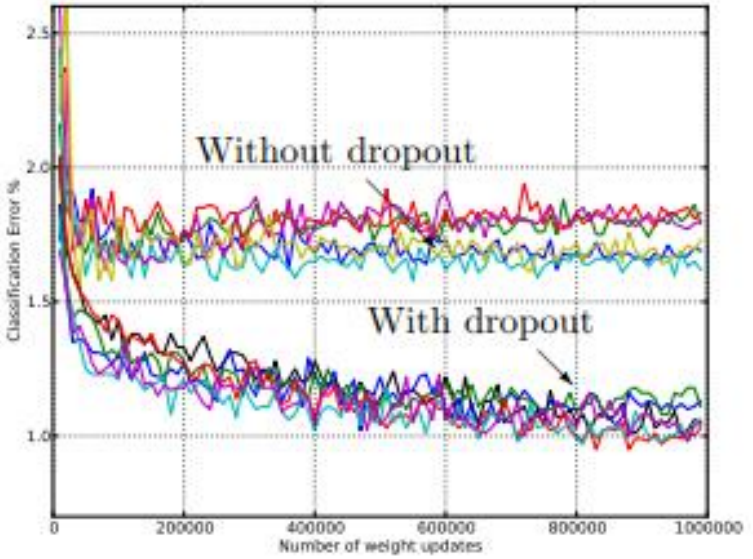


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

6.1.2 Street View House Numbers

The Street View House Numbers (SVHN) Data Set (Netzer et al., 2011) consists of color images of house numbers collected by

Paper Name: Dropout: A Simple Way to Prevent Neural Networks from Overfitting