SMOTE and Metric Learning

KLINGELHOEFER Felix

Febuary 2019

1 Introduction

During my internship, I studied the Synthetic Minority Over-sampling TEchnique (SMOTE), as well as Metric Learning, in an effort to combine them and design a novel metric learning method for imbalanced datasets. I worked with a PhD student who was developing a new Metric Learning method based on the Mahalanobis distance, for imbalanced datasets. The goal was to use the distance that was learned as an alternative to the Euclidean distance in the SMOTE algorithm, and try to learn the optimal placement for synthetic points.

2 The Data Imbalance Problem

In many machine learning tasks, one class is much more represented than another: be it fraud, fault or disease detection, the class of objects that needs to be detected is often quite rare. This can spell trouble for many classic machine learning problems: faced with an overwhelming majority of points belonging to one class, a classifier that is trained to increase accuracy will tend to predict all points as belonging to that class.

2.1 Algorithmic methods

There are several ways of dealing with the class imbalance problem. First of all, some machine learning methods are trained to optimize more appropriate measures than accuracy, which are derived from the confusion matrix.

		Predicted Class	
		1	**0**
Actual Class	**1**	True Positive	False Negative
	0	False Positive	True Negative

We can then define the following measures, by using the number of True Positives (tp), False Positives (fp), True Negatives (tn) and False Negatives (fn).

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \tag{1}$$

$$\text{Precision} = \frac{tp}{tp + fp} \tag{2}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{3}$$

$$F_1 = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

The F_1 measure is often used when datasets are imbalanced, because it doesn't favor the majority class like Accuracy. Another algorithmic solution to the class imbalance problem is to give different weights to majority and minority points during training. In a nutshell, the cost function is adapted so that an error on a minority point is as costly as many errors on majority class points. This ratio can be adapted to the imbalance in the data, and the preferred optimization (maximizing precision or recall).

2.2 Sampling methods

If the dataset is very large, a very simple and often very efficient way of dealing with class imbalance is to undersample the majority class. The most basic undersampling method is to randomly select a proportion of the majority class points, but this can lead to important data points being lost. Using clustering methods (such as K-Means) can usually overcome this risk when enough data is present.

Undersampling is usually the preferred method when the size of the dataset is prohibitive for training the machine learning algorithm, or in order to improve efficiency. In other cases, oversampling the minority class will lead to greater diversity in the data, so usually to better results for algorithms trained with it. As for undersampling, the most basic method is random oversampling: points are randomly selected from the original minority class and duplicated, until the desired balance is reached. More efficient oversampling methods have been developed however, and they are nearly all based on the Synthetic Minority Oversampling TEchnique (SMOTE)[CBHK02].

2.3 SMOTE and variants

One of the drawbacks of random oversampling is that points are repeated, which can lead to a very narrow decision region for the minority class. The idea of SMOTE is to increase diversity by generating synthetic points that are very likely to belong to the minority class, but are not yet present in the dataset.

In order to do so, synthetic points are generated by selecting the k nearest same-class neighbors of every minority point (through euclidean distance), and then randomly choosing one of them, and adding a new point whose features are all randomly chosen in between the values of the features of its two parents. This means the point is randomly placed in the n-dimensional box between the two points it is generated from. Although

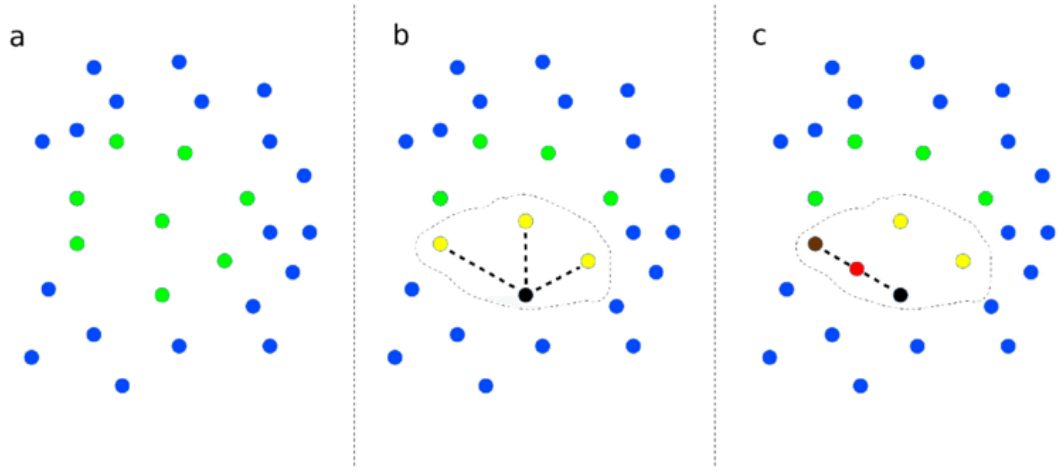

Figure 1: Graphical representation of SMOTE. a: minority points (green) and majority points (blue) are separated b: for each minority point, its k nearest minority neighbors are selected (here k=3) c: One of those neighbors is chosen randomly, the synthetic point is then generated randomly between both points (for each feature)

this is clear in the papers by the original authors, there is some confusion in many derived methods, as well as in the implementations of some well-known libraries (such as sklearn), who place the point on the line between both parents (as in Figure 1).

Many methods have improved upon SMOTE, by either selecting points based on certain conditions (Borderline SMOTE[HWM05], or ADASYN[HBGL08]), or combining it with different methods (boosting for SMOTEboost [HBGL08]). Our idea was to combine SMOTE with metric learning, and try to learn the position of the point between its parents, instead of selecting it randomly.

3 Metric Learning

Metric learning [BHS15, Kul13], a subfield of representation learning, consists of designing a pairwise function able to measure the dis/similarity between two data points. This issue is key in machine learning where such metrics are at the core of many algorithms, like K-nearest neighbors (KNN), SVMs, K-Means, etc. To construct a dis/similarity measure suitable for a given task, most metric learning algorithms optimize a loss function which aims at bringing closer examples of the same label while pushing apart examples of different labels. In practice, metric learning is usually performed with cannot link/must link constraints—x and x' should be dis/similar [XJRN03, DKJ$^+$07, WS09, XNZ08, LZT$^+$14, ZHS16]—or relative constraints—x should be more similar to x' than to x'' [SJ04, LJJ08, WS09, ZGX11].

We focused on the family of metric learning algorithms that construct a Mahalanobis distance $d_{\mathbf{M}}(x, x') = \sqrt{(x - x')^\top \mathbf{M}(x - x')}$ parameterized by a positive semidefinite matrix $\mathbf{M}$. Learning a Mahalanobis distance leads to several nice properties: *(i)* $d_{\mathbf{M}}$ is a generalization of the Euclidean distance; *(ii)* it induces a projection such that the

distance between two points is equivalent to their Euclidean distance after a linear projection; *(iii)* $\mathbf{M}$ can be low rank implying a projection in a lower dimensional space; *(iv)* it involves optimization problems that are often convex and thus easy to solve. The most famous Mahalanobis distance learning algorithms are likely, **LMNN** (Large Margin Nearest Neighbor [WS09]) and **ITML** (Information-Theoretic Metric Learning [DKJ$^+$07]), which are both designed to improve the accuracy of the KNN classification rule in the latent space. The principle of **LMNN** is the following: for each training example, its K nearest neighbors of the same class (the *target neighbors*) should be closer than instances of other classes (the *impostors*). **ITML** uses a LogDet regularization and minimizes (*resp.* maximizes) the distance between examples of the same (*resp.* different) class. The loss functions optimized in **LMNN** and **ITML** (and in most pairwise metric learning methods) tend to favor the majority class as there is no distinction between the constraints involving examples of the majority class and the constraints on the minority class. This strategy is thus, as explained previously, not well suited when dealing with imbalanced datasets. An illustration of this phenomenon on the BALANCE dataset from the UCI repository is shown in Figure **??**. We observe that increasing the imbalance ratio tends to generate a metric which classifies (with a KNN rule) all the examples as the majority class, thus leading to an accuracy close to 1. On the other hand, the F-measure decreases with the proportion of positives, showing that the classifier missed many positives, usually considered as the examples of interest.

3.1 Imbalanced Metric Learning

The PhD Student I was working with developed a Metric Learning algorithm that I based my work on. The following is an extract of his manuscript presenting the method.

Classic Mahalanobis metric learning algorithms [JWZ09, BHS15, CGY16] are usually expressed as

$$\min_{\mathbf{M} \succeq 0} F(\mathbf{M}) = \frac{1}{n^2} \sum_{(z_i, z_j) \in \mathcal{S}^2} \ell(\mathbf{M}, z_i, z_j) + \lambda Reg(\mathbf{M}), \tag{5}$$

where one wants to minimize the trade-off between a convex loss ℓ over all pairs of examples and a regularization Reg under the PSD constraint $\mathbf{M} \succeq 0$.

The major drawback of this formulation is that the loss gives the same importance to any pair of examples (z_i, z_j) whatever their label y_i and y_j. Intuitively, this is indeed not well suited to imbalanced data where the minority class is the set of examples of interest. Some metric learning algorithms [WS09, ZHS16] allow to weight the role played by the must-link and cannot-link constraints. However, the problem still holds because the labels of the examples are not directly taken into account.

Our simple idea is to decompose further the sets of pairs of examples based on their labels. Each set can then be associated to a specific weight during the optimization to reduce the negative effect of the imbalance. Concretely, we propose to decompose the

sum over all pairs of Equation (5) into four terms:

$$\min_{\mathbf{M}\succeq 0} F(\mathbf{M}) = \frac{1}{n^2}\Bigg(\sum_{(z_i,z_j)\in Sim^+} \ell(\mathbf{M}, z_i, z_j) + \sum_{(z_i,z_j)\in Dis^+} \ell(\mathbf{M}, z_i, z_j) + \sum_{(z_i,z_j)\in Dis^-} \ell(\mathbf{M}, z_i, z_j) + \sum_{(z_i,z_j)\in Sim^-} \ell(\mathbf{M}, z_i, z_j) \Bigg) + \lambda Reg(\mathbf{M}), \tag{6}$$

where the four sets Sim^+, Dis^+, Dis^- and Sim^- are defined as subsets of $\mathcal{S}\times\mathcal{S}$ respectively as: $Sim^+\subseteq\mathcal{S}^+\times\mathcal{S}^+$, $Dis^+\subseteq\mathcal{S}^+\times\mathcal{S}^-$, $Dis^-\subseteq\mathcal{S}^-\times\mathcal{S}^+$ and $Sim^-\subseteq\mathcal{S}^-\times\mathcal{S}^-$. We set the regularization term as $Reg(\mathbf{M}) = \|\mathbf{M}-\mathbf{I}\|_F^2$ where $\mathbf{I}$ is the identity matrix and $\|.\|_F$ is the Frobenius norm. It aims at avoiding over-fitting by enforcing $\mathbf{M}$ to be close to the identity matrix $\mathbf{I}$.

The importance of each of the four sums can then be incorporated into the loss function ℓ that we define as follows: $\forall\ (z, z') \in \mathcal{Z}^2$

$$\ell(\mathbf{M}, z, z') = \begin{cases} a\ell_1(\mathbf{M}, x, x') & \text{if } y{=}{+}1, y'{=}{+}1, \\ b\ell_2(\mathbf{M}, x, x') & \text{if } y{=}{+}1, y'{=}{-}1, \\ c\ell_2(\mathbf{M}, x, x') & \text{if } y{=}{-}1, y'{=}{+}1, \\ d\ell_1(\mathbf{M}, x, x') & \text{if } y{=}{-}1, y'{=}{-}1, \end{cases} \tag{7}$$

with $\ell_1(\mathbf{M}, x, x') = [d^2_{\mathbf{M}}(x, x') - 1]_+$ and $\ell_2(\mathbf{M}, x, x') = [1 + m - d^2_{\mathbf{M}}(x, x')]_+$ where $[.]_+$ is the Hinge loss and $m \geq 0$ a margin parameter. The idea of ℓ_1 is to bring examples of the same class at a distance less than 1 while ℓ_2 aims to push far away examples of different classes at a distance larger than 1 plus a given margin m.

If we look more closely at the proposed Equation (6), when all pairs from $\mathcal{S}\times\mathcal{S}$ are involved, Sim^+ and Sim^- contain respectively n^+n^+ and n^-n^- pairs while Dis^+ and Dis^- contain respectively n^+n^- and n^-n^+ elements. This means that Dis^+ and Dis^- contain the symmetric pairs and might be merged. However, metric learning methods rarely consider all the possible pairs as it becomes quite inefficient in the presence of a large number of examples. Possible strategies to select the pairs include a random subsampling [XJRN03, DKJ+07, XNZ08, ZHS16] or a selection based on the nearest neighbors rule [WS09, LZT+14]. For this reason, it might make sense to separate the two sets Dis^+ and Dis^- and allow us to weight them differently as they may not consider the same subsets of pairs. Another interpretation of such a decomposition in an imbalanced learning setting is the following: if z_j is selected as belonging to the neighborhood of z_i, the minimization of the four terms of Equation (6) can be seen as a nice way to implicitly optimize with a KNN rule the true positive, false negative, false positive and true negative rates respectively.

4 SMOTE and IML combination

4.1 Theoretical framework

The original idea to combine metric learning and SMOTE was to do a dual gradient descent, alternating between learning the mahalanobis projection as in the IML method, and learning the position of the synthetic points. In order to do so, synthetic points were placed on a line between two parent points, x and y. Each synthetic point s_i therefore had coordinates $x_i + \alpha_i * (y_i - x_i)$. The loss function, which was the same as the loss function for IML, was then differentiated in function of the α_i in order to learn the optimal position for each synthetic point between its parents. Mathematically, we had a set E, of elements with label function $l : E \to \{-1, 1\}$ such that:

$$\begin{aligned} np &= |\{e \in E | l(e) = 1\}| \\ nn &= |\{e \in E | l(e) = -1\}| \\ np &< nn \\ ns &= \lfloor \frac{nn}{np} + 1 \rfloor * np \end{aligned} \tag{8}$$

In other words, np is the number of points belonging to the majority class, nn the number of points of the minority class, and ns the number of generated synthetic points, which leads to approximate balance. The loss function is then :

$$\begin{aligned} f(\alpha, x, y, p, n, M) =& a \sum_{i=1}^{ns} \alpha_i(1 - \alpha_i) \\ &+ b \sum_{i=1}^{ns*np} [(x_i + \alpha_i(y_i - x_i) - p_i)^T * M * (x_i + \alpha_i(y_i - x_i) - p_i) - 1] \\ &+ c \sum_{i=1}^{ns*nn} [1 + m - (x_i + \alpha_i(y_i - x_i) - n_i)^T * M * (x_i + \alpha_i(y_i - x_i) - n_i)] \\ &+ d \sum_{i=1}^{np*np} [(p_i - p_i')^T * M * (p_i - p_i') - 1] \\ &+ e \sum_{i=1}^{nn*nn} [(n_i - n_i')^T * M * n_i - n_i') - 1] \\ &+ f \sum_{i=1}^{ns*nn} [1 + m - (p_i - n_i)^T * M * (p_i - n_i)] \\ &+ g * ||M - I||_F^2 \end{aligned} \tag{9}$$

The $\alpha_i(1 - \alpha_i)$ term is a regularization for the synthetic points to not be too far from the center of the line between its parents. There is no guarantee that $0 < \alpha_i < 1$ so a synthetic point generated from x and y could be placed outside of the $[x, y]$ segment, but will always be on the (x, y) line. Synthetic points were originally considered the same as minority class points, but experimentation showed that making them separate in the

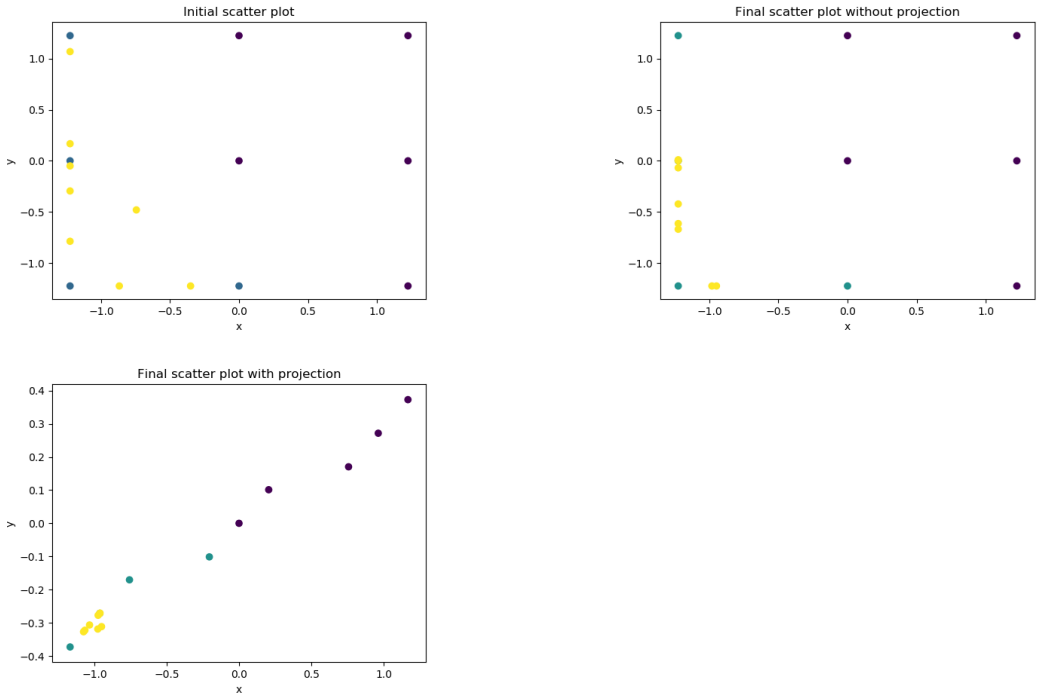

Figure 2: Plots of the lMLS algorithm on a simple 2d example. Purple: Majority points, Blue: Minority points, Yellow: Synthetic points

loss function, and removing the term that would bring synthetic points closer to other synthetic points, led to better results. The differentiation for each α_i of this loss function is then:

$$\begin{aligned}\frac{\delta f(\alpha, x, y, p, n)}{\delta \alpha_i} =& a(1 - 2\alpha_i) \\ &+ b[2 * (x_i + \alpha_i(y_i - x_i) - p_i)^T * M * (x_i - y_i)] \\ &+ c[-2 * (x_i + \alpha_i(y_i - x_i) - n_i)^T * M * (x_i - y_i)]\end{aligned} \tag{10}$$

This gradient descent on the position of the points can also be performed with other methods, and I later tried combining this learning on SMOTE with other algorithms.

4.2 Experimentation

In our experiments, we compared several algorithms with the learned SMOTE, the basic smote and no oversampling on different imbalanced datasets. The metric learning algorithms used were **IML**, **LMNN**, **Euclidean** and **MLS**. The two first were described previously, **Euclidean** is just using the Euclidean distance (instead of a learned one) during classification, and **MLS** is the variant of **ITML** with the loss function we developed that discriminates synthetic and minority points.

For our comparison, we did a 5-fold cross-validation of the hyperparameters with 20 different configurations for all the methods on the training set (using 4/5ths of the training set to learn and 1/5th to validate), and then used the best hyperparamaters to learn on the entire training set, and test on the rest of the dataset. After learning a

Dataset	lMLS	MLS	IML	oIML	LMNN	oLMNN	lLMNN	Euclidean	oEuclidean	lEuclidean
glass	69.4 ± 4.7	68.2 ± 3.8	69.2 ± 9.4	67.7 ± 6.5	67.6 ± 8.6	68.6 ± 6.3	67.8 ± 5.6	67.5 ± 6.7	69.0 ± 5.2	$\mathbf{70.3} \pm 4.5$
hayes	84.1 ± 10.3	86.1 ± 8.7	71.2 ± 14.7	79.8 ± 11.0	74.0 ± 12.7	86.9 ± 10.4	85.5 ± 11.7	63.6 ± 14.2	85.9 ± 9.0	$\mathbf{88.5} \pm 8.2$
libras	85.3 ± 8.7	$\mathbf{87.5} \pm 9.4$	80.6 ± 12.4	85.4 ± 9.6	82.6 ± 10.8	87.3 ± 10.4	84.5 ± 8.7	74.6 ± 10.3	77.5 ± 10.6	78.9 ± 8.8
newthyroid	$\mathbf{94.4} \pm 3.7$	93.7 ± 3.6	92.8 ± 4.1	92.2 ± 4.4	85.1 ± 5.6	88.1 ± 6.3	90.8 ± 4.7	85.4 ± 5.7	86.9 ± 4.4	92.0 ± 3.3
spectfheart	45.7 ± 5.1	45.3 ± 5.6	28.3 ± 11.7	42.7 ± 6.0	41.9 ± 10.0	46.2 ± 7.9	50.5 ± 6.8	37.0 ± 9.5	$\mathbf{50.5} \pm 5.3$	50.5 ± 4.7
wine	$\mathbf{98.2} \pm 2.6$	98.1 ± 2.8	98.1 ± 2.5	97.8 ± 2.3	96.6 ± 3.0	95.8 ± 3.8	89.3 ± 4.0	86.0 ± 3.4	85.7 ± 3.7	93.7 ± 3.5
Mean	79.5 ± 5.9	$\mathbf{79.8} \pm 5.7$	73.4 ± 9.1	77.6 ± 6.6	74.6 ± 8.5	78.8 ± 7.5	78.1 ± 6.9	69.0 ± 8.3	75.9 ± 6.4	79.0 ± 5.5
Total time	570 min	40 min	17 min	25 min	192 min	220 min	764 min	0 min	0 min	520 min
Average Rank	3.00	3.14	6.14	5.57	7.71	4.29	5.43	9.57	6.14	4.00

Table 1: Results of our experiments on different datasets. l...: method with learned SMOTE, o...: method with basic SMOTE. Scores averaged over 40 iterations, with mean results and standard deviation indicated.

projection with the metric learning algorithms, classification was done using the K-NN algorithm. We give the results of our experiments in Figure .

Before applying the metric learning algorithms, the data was normalized on each axis. In the experiments of Figure , however, this normalization was not done for the Euclidean and oEuclidean methods. Later tests showed that the learned smote combined with the euclidean method were actually slightly less good than the euclidean with basic SMOTE. Taking this into account, we can say that learning the position of the points actually led to worse results in combination with any of the metric learning techniques. This seemed very surprising, which led us to do more experiments comparing smote and learned SMOTE, which is when we realized that SMOTE actually placed points in the n-orthotope between parent points and not the line. Experiments showed that the learned SMOTE performed nearly exactly the same as the SMOTE with a line, both slightly less well than the true SMOTE. This still indicates that our learning on the position of the points had no benefits.

We did however manage to improve the results of the **IML** method combined with SMOTE, by making the algorithm distinguish minority points from synthetic points. This leads us to believe that a good way of improving SMOTE may be by adapting subsequent algorithms that receive the oversampled data. While this is not an easily generalizable method, it does raise interesting perspectives for the field of oversampling.

Many other attempts were made to improve the learning, for example alternating the gradient descents to learn the positions and the projection instead of doing one completely and then the other, or changing the loss function to a more adversarial view, trying to place the synthetic points closer to the majority class points to provide more diversity. Sadly, none of them led to a noticeable gain in performance. In our choice of parent points for the SMOTE algorithm, we also tried to use the learned Mahalanobis distance instead of the basic Euclidean distance (in the choice of the K nearest neighbors), but that did not change performances either.

5 Conclusion

We built upon a metric learning method and tried to combine it with SMOTE, in order to learn the position of the points in the SMOTE algorithm. While we were unsuccessful in our attempts, we did highlight an interesting discrepancy between the original SMOTE

and most implementations and further extensions of the method (placing points in the n-orthotope instead of the line). We also managed to improve the results of the metric learning algorithm by making it sensitive to the difference between synthetic and minority points, which may be interesting for the field of oversampling : adapting other algorithms to learn on oversampled data may also lead to performance improvements. While our attempts at introducing learning into the SMOTE algorithms did not provide meaningful results, maybe developing a new oversampling method from ground-up around a learning method could prove more successful. We can cite the conditional Generative Adversarial Networks (cGAN) that have showed much promise in that field recently[DB17].

References

[BHS15] A. Bellet, A. Habrard, and M. Sebban. Metric learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 9(1), 2015.

[CBHK02] N. Chawla, K. Bowyer, L. Hall, and P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[CGY16] Q. Cao, Z. Guo, and Y. Ying. Generalization bounds for metric and similarity learning. *Machine Learning*, 102(1):115–132, 2016.

[DB17] Georgios Douzas and Fernando Bação. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with Applications*, 91, 09 2017.

[DKJ+07] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon. Information-theoretic metric learning. In *ICML*, 2007.

[HBGL08] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008.

[HWM05] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I*, ICIC'05, pages 878–887, Berlin, Heidelberg, 2005. Springer-Verlag.

[JWZ09] R. Jin, S. Wang, and Y. Zhou. Regularized distance metric learning: Theory and algorithm. In *NIPS*, 2009.

[Kul13] B. Kulis. Metric learning: A survey. *Foundations and Trends in ML*, 5(4):287–364, 2013.

[LJJ08] J. Lee, R. Jin, and A. Jain. Rank-based distance metric learning: An application to image retrieval. In *CVPR*, 2008.

[LZT+14] J. Lu, X. Zhou, Y. Tan, Y. Shang, and J. Zhou. Neighborhood repulsed metric learning for kinship verification. *IEEE transactions on pattern analysis and machine intelligence*, 36(2):331–345, 2014.

[SJ04] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2004.

[WS09] K. Weinberger and L. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10(Feb):207–244, 2009.

[XJRN03] E. Xing, M. Jordan, S. Russell, and A. Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, 2003.

[XNZ08] S. Xiang, F. Nie, and C. Zhang. Learning a mahalanobis distance metric for data clustering and classification. *Pattern Recognition*, 41(12):3600–3612, 2008.

[ZGX11] W. Zheng, S. Gong, and T. Xiang. Person re-identification by probabilistic relative distance comparison. 2011.

[ZHS16] P. Zadeh, R. Hosseini, and S. Sra. Geometric mean metric learning. In *ICML*, 2016.