AP-Perf: Incorporating Generic Performance Metrics in Differentiable Learning

RIZAL FATHONY AND ZICO KOLTER (1),(2)

(1) Carnegie Melon University
(2) Bosch Center for AI

Highlight of our paper

A generic framework and programming tools
that enable practitioners to easily
align training objective with the evaluation metric

Integration with ML Pipelines

Easily incorporates custom performance metrics into machine learning pipeline

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(30, 100)
        self.fc2 = nn.Linear(100, 100)
        self.fc3 = nn.Linear(100, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x).squeeze()
```

```
model = Net().to(device)
criterion = nn.BCEWithLogitsLoss().to(device)
```

```
optimizer.zero_grad()
objective = criterion(model(inputs), labels)
objective.backward()
optimizer.step()
```

Leaning using binary cross entropy

*) Code in PyTorch

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(30, 100)
        self.fc2 = nn.Linear(100, 100)
        self.fc3 = nn.Linear(100, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x).squeeze()
```

```
class FBeta(PerformanceMetric):
    def __init__(self, beta):
        self.beta = beta

    def define(self, C: Confusion_Matrix):
        return ((1 + self.beta ** 2) * C.tp) \
                / ((self.beta ** 2) * C.ap + C.pp)

f2_score = FBeta(2)
f2_score.initialize()
f2_score.enforce_special_case_positive()

model = Net().to(device)
criterion = MetricLayer(f2_score).to(device)
```

```
optimizer.zero_grad()
objective = criterion(model(inputs), labels)
objective.backward()
optimizer.step()
```

Leaning using AP formulation for F2-metric

F_β score definition

$$\frac{(1+\beta^2)\ \text{TP}}{\beta^2\ \text{AP} + \text{PP}}$$

Motivation

Evaluation Metrics

Example: Digit Recognition

Evaluation Metric:

Performance Metric: Accuracy

$$\text{Accuracy} = \frac{\text{\# correct prediction}}{\text{\# sample}}$$

Loss Metric: Zero-One Loss

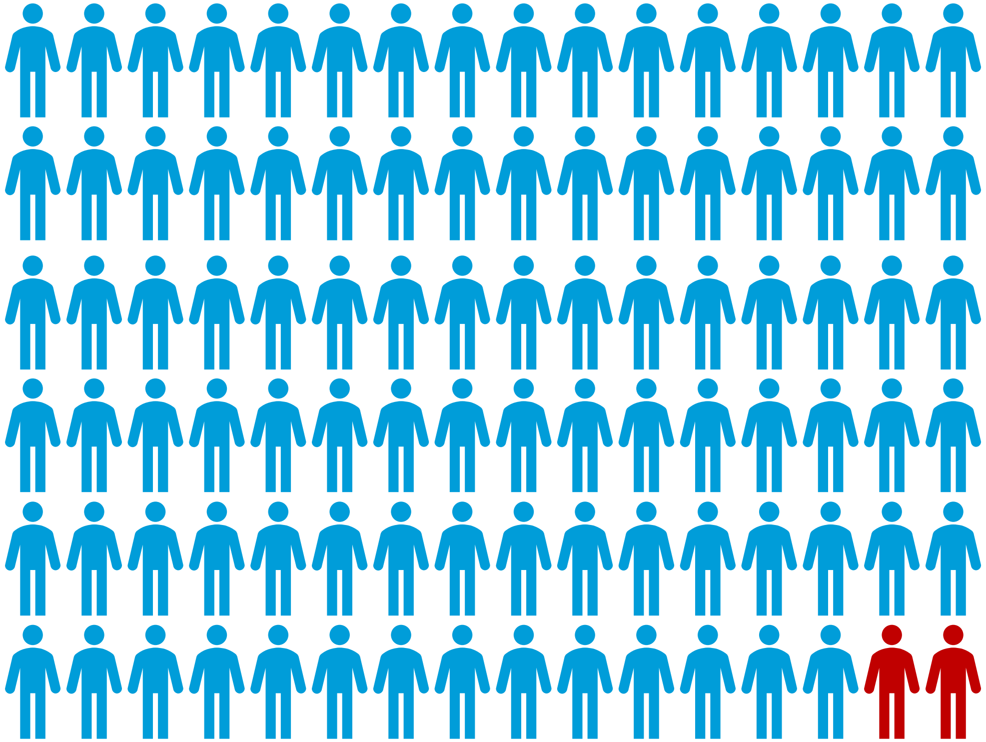
$$\text{Zero-One Loss} = \frac{\text{\# incorrect prediction}}{\text{\# sample}}$$

Most widely used metric!

Accuracy metric is not always desirable

Example: Disease Prediction

(imbalanced dataset)



98% of the samples: healthy (negative samples)

2% of the samples: have disease (positive samples)

Predict all samples as negative:

Accuracy metric: 98%

Confusion Matrix

		Actual		
		Positive	Negative	
Pred.	Positive	True Pos. (TP)	False Pos. (FP)	Predicted Pos. (PP)
	Negative	False Neg. (FN)	True Neg. (TN)	Predicted Neg. (PN)
		Actual Pos. (AP)	Actual Neg. (AN)	All Data (ALL)

$$\text{Precision} = \frac{\#\ \text{true positive}}{\#\ \text{predicted positive}} \qquad \text{Recall} = \frac{\#\ \text{true positive}}{\#\ \text{actual positive}}$$

$$\text{Specificity} = \frac{\#\ \text{true negative}}{\#\ \text{actual negative}} \qquad \text{Sensitivity} = \frac{\#\ \text{true positive}}{\#\ \text{actual positive}}$$

$$\text{F1-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$F_\beta\text{-score} = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

Learning Tasks & Evaluation Metrics

Machine Learning Tasks	Popular Evaluation Metrics
Imbalanced Datasets	- F1-Score - Area under ROC Curve (AUC) - Precision vs Recall
Medical classification tasks	- Specificity - Sensitivity - Bookmaker Informedness
Information retrieval tasks	- Precision@k - Mean Average Precision (MAP) - Discounted cumulative gain (DCG)
Weighted classification tasks	- Cost-sensitive loss metric
Rating tasks	- Cohen’s kappa score - Fleiss' kappa score
Computational biology tasks	- Precision-Recall curve - Matthews correlation coefficient (MCC)

Evaluation Metric vs Training Model Mismatch

Example: Disease prediction

Evaluation Metric

vs

ML model

Training objective

Optimize specificity & sensitivity

Most of ML models:
- No support for specificity & sensitivity metric
- Optimize the cross-entropy objective
  (a proxy for accuracy metric)

Discrepancy:
Evaluation metrics vs training objective

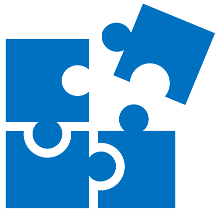
Inferior performance results
(Cortes & Mohri, 2004; Eban et.al, 2016)

Our paper

A generic framework and programming tools
that enable practitioners to easily
align training objective with the evaluation metric

Related Works

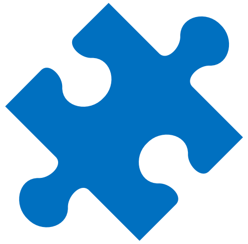
Evaluation Metrics



Decomposable Metrics

Can be decomposed into sample-wise sum

Example: accuracy, ordinal regression, and cost-sensitive metrics.



Non-Decomposable Metrics

Cannot be decomposed into sample-wise sum

Example: F1-score, GPR, informedness, MCC, Kappa score.

Common in many applications

Learning Algorithm Design

Empirical Risk Minimization Framework:
Approximate the evaluation metrics (discrete, non-continuous) with convex surrogate losses.

Binary classification | accuracy

Evaluation Metric:

Accuracy metric

Convex Surrogate Losses

- ✓ Hinge Loss :: Support Vector Machine
- ✓ Log Loss :: Logistic Regression
- ✓ Exponential Loss :: AdaBoost

Non-decomposable metrics

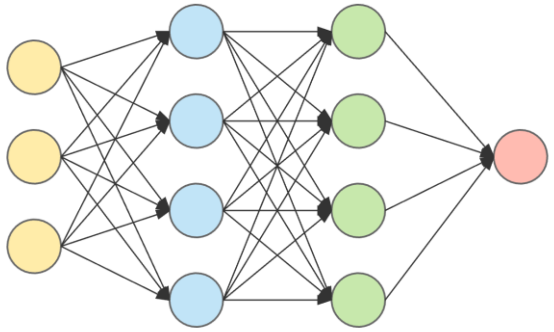
SVM-based model: SVM-perf (Joachims, 2005)

- ✓ Works on many complex metrics
- ✗ No statistical consistency guarantee
- ✗ Does not provide easy tool to extend the method to custom metrics

Most of other models:
(e.g.: Koyejo et al, 2014; Narashiman et al, 2014)

- ✗ Hard to extend to custom metrics

Neural Networks Learning



Currently the popular machine learning model.

Use the classical surrogate losses as the last layer (objective).

Classification with accuracy metric

Objective: Cross entropy objective

= Logistic regression (log-loss surrogate)

Non-decomposable metrics

Most of 'classical' models:

✗ Not applicable to NN learning

NN-targeted models:
(Eban et.al, 2016; Song et.al, 2016; Sanyal, et.al, 2018)

✗ Only support few metrics

✗ No support for custom metrics

Practitioners' perspective

Aim to optimize an evaluation metric tailored specifically for their problem.
(e.g. specificity, sensitivity, kappa score)

No learning models can easily optimize their specific evaluation metrics.

Choose the standard cross entropy instead

Mismatch between
Evaluation Metric vs Training Model

Approach

Adversarial Prediction (Asif et.al, 2016; Fathony et.al, 2018)

Empirical Risk Minimization

Optimize Original Evaluation Metric

Discrete, Intractable

Approximate the metric

Exact training data

Empirical Risk Minimization with Convex Surrogate Loss

Convex, Tractable

More complex metric →
Harder to construct good surrogate losses

Exact evaluation metric

Approximate training data

Adversarial Prediction

(Asif et.al ’16; Fathony et.al, ‘18)

Convex, Tractable

Adversarial Prediction

No need to independently construct surrogate loss for every metric

Our Method

OPTIMIZING GENERIC NON-DECOMPOSABLE METRICS

Non-Decomposable Metric

		Actual		
		Positive	Negative	
Pred.	Positive	True Pos. (TP)	False Pos. (FP)	Predicted Pos. (PP)
Pred.	Negative	False Neg. (FN)	True Neg. (TN)	Predicted Neg. (PN)
		Actual Pos. (AP)	Actual Neg. (AN)	All Data (ALL)

Example:

Binary Classification with F1-score metric

$$\text{F1-score}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\ \text{TP}}{\text{PP} + \text{AP}} = \frac{2 \sum_i \hat{y}_i y_i}{\sum_i \hat{y}_i + \sum_i y_i}$$

AP | Decomposable metric (Asif et.al, 2015; Fathony et.al, 2016, 2017, 2018) :

Reduces to an optimization over sample-wise conditional probability distributions.

$\mathcal{P}(\hat{y}_i|\mathbf{x}_i)$
Size: 2 (binary) x # sample

AP | Non-decomposable metric:

Requires optimization over full training set conditional probability distribution

$\mathcal{P}(\hat{\mathbf{y}}|\mathbf{x})$
Size: 2^n (exponential)

Marginalization technique: optimize over marginalization distribution instead:

Original: $\mathcal{P}(\hat{\mathbf{y}}|\mathbf{x})$
Size: 2^n
Intractable!

→ Marginalization: $\mathcal{P}(\hat{y}_i = 1, \sum_i \hat{y}_i = k|\mathbf{x})$
Size: n^2 Tractable!

Generic Non-Decomposable Performance Metrics

More complex performance metric

$$\text{metric}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_j \frac{a_j \mathsf{TP} + b_j \mathsf{TN} + f_j(\mathsf{PP}, \mathsf{AP})}{g_j(\mathsf{PP}, \mathsf{AP})}$$

		Actual Positive	Actual Negative	
Pred.	Positive	True Pos. (TP)	False Pos. (FP)	Predicted Pos. (PP)
Pred.	Negative	False Neg. (FN)	True Neg. (TN)	Predicted Neg. (PN)
		Actual Pos. (AP)	Actual Neg. (AN)	All Data (ALL)

Cover a vast range of performance metric families

Including most common use cases of non-decomposable metrics:
Precision, Recall, F_β-score, Balanced Accuracy, Specificity, Sensitivity, Informednes, Markedness, MCC, Kappa score, etc…

Practitioners can define their novel custom metrics

Metrics that specifically targeted to their novel problems.

Marginalization technique:

Original: $\mathcal{P}(\hat{\mathbf{y}}|\mathbf{x})$ Size: 2^n Intractable! → Marginalization: $\mathcal{P}(\hat{y}_i = 1, \sum_i \hat{y}_i = k|\mathbf{x})$ & $\mathcal{P}(\hat{y}_i = 0, \sum_i \hat{y}_i = k)$ Size: $2n^2$ Tractable!

Optimization: Gradient Descent + an ADMM-based solver (inner optimization)

Integration with ML Pipelines

Easily incorporates custom performance metrics into machine learning pipeline

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(30, 100)
        self.fc2 = nn.Linear(100, 100)
        self.fc3 = nn.Linear(100, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x).squeeze()

model = Net().to(device)
criterion = nn.BCEWithLogitsLoss().to(device)

optimizer.zero_grad()
objective = criterion(model(inputs), labels)
objective.backward()
optimizer.step()
```

Leaning using binary cross entropy

*) Code in PyTorch

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(30, 100)
        self.fc2 = nn.Linear(100, 100)
        self.fc3 = nn.Linear(100, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x).squeeze()

class FBeta(PerformanceMetric):
    def __init__(self, beta):
        self.beta = beta

    def define(self, C: Confusion_Matrix):
        return ((1 + self.beta ** 2) * C.tp) \
            / ((self.beta ** 2) * C.ap + C.pp)

f2_score = FBeta(2)
f2_score.initialize()
f2_score.enforce_special_case_positive()

model = Net().to(device)
criterion = MetricLayer(f2_score).to(device)

optimizer.zero_grad()
objective = criterion(model(inputs), labels)
objective.backward()
optimizer.step()
```

Leaning using AP formulation for F2-metric

F_β score definition

$$\frac{(1+\beta^2)\ \mathrm{TP}}{\beta^2\ \mathrm{AP} + \mathrm{PP}}$$

AP-Perf: supports a wide variety of evaluation metrics

Code examples for other performance metrics:

Geometric Mean of Precision and Recall (GPR)

$$\frac{\mathsf{TP}}{\sqrt{\mathsf{PP} \cdot \mathsf{AP}}}$$

```
class GM_PrecRec(PerformanceMetric):
    def define(self, C: Confusion_Matrix):
        return C.tp / sqrt(C.ap * C.pp)

gpr = GM_PrecRec()
gpr.initialize()
gpr.enforce_special_case_positive()
```

Cohen's Kappa score

$$\frac{(\mathsf{TP} + \mathsf{TN})/\mathsf{ALL} - (\mathsf{AP} \cdot \mathsf{PP} + \mathsf{AN} \cdot \mathsf{PN})/\mathsf{ALL}^2}{1 - (\mathsf{AP} \cdot \mathsf{PP} + \mathsf{AN} \cdot \mathsf{PN})/\mathsf{ALL}^2}$$

```
class Kappa(PerformanceMetric):
    def define(self, C: Confusion_Matrix):
        pe = (C.ap * C.pp + C.an * C.pn) / C.all**2
        num = (C.tp + C.tn) / C.all - pe
        den = 1 - pe
        return num / den

kappa = Kappa()
kappa.initialize()
kappa.enforce_special_case_positive()
kappa.enforce_special_case_negative()
```

Novel Custom Metrics

Write-your-own Novel Metrics

```python
class NovelMetric(PerformanceMetric):
    def define(self, C: Confusion_Matrix):
        # write the definition of your new metric
```

Example:

a weighted modification to the Cohen's Kappa score and the Mathews correlation coefficient (MCC)

$$0.3 \cdot \frac{(0.7\ \mathsf{TP} + 0.3\ \mathsf{TN})\,/\,\mathsf{ALL} - (0.7 \cdot \mathsf{AP} \cdot \mathsf{PP} + 0.3 \cdot \mathsf{AN} \cdot \mathsf{PN})\,/\,\mathsf{ALL}^2}{1 - (0.7 \cdot \mathsf{AP} \cdot \mathsf{PP} + 0.3 \cdot \mathsf{AN} \cdot \mathsf{PN})\,/\,\mathsf{ALL}^2} + 0.7 \cdot \frac{\mathsf{TP}\,/\,\mathsf{ALL} - (\mathsf{AP} \cdot \mathsf{PP})\,/\,\mathsf{ALL}^2}{\sqrt{\mathsf{AP} \cdot \mathsf{PP} \cdot \mathsf{AN} \cdot \mathsf{PN}}\,/\,\mathsf{ALL}^2}$$

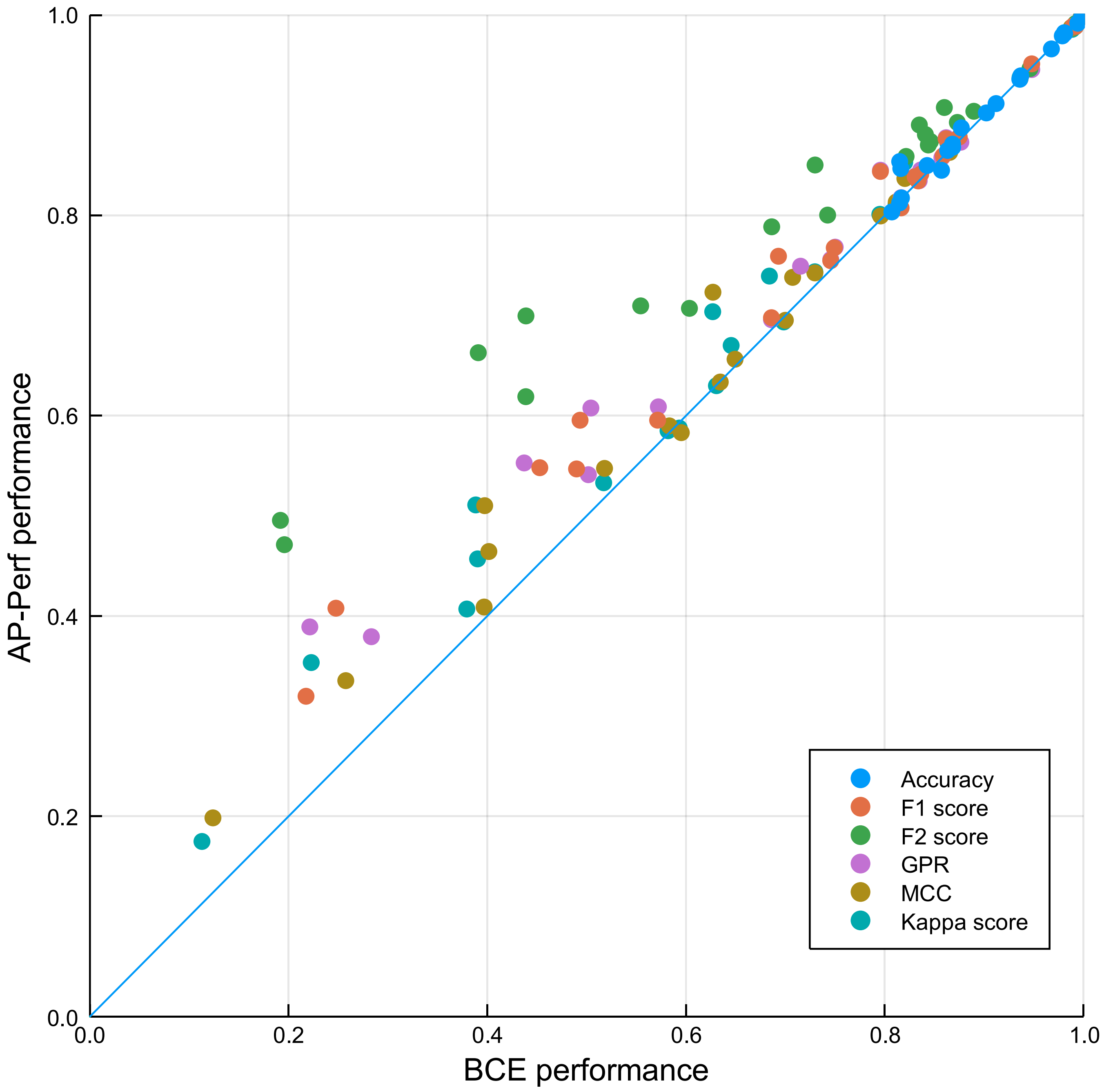
```python
class NovelMetric(PerformanceMetric):
    def define(self, C: Confusion_Matrix):
        pe = (0.7 * C.ap * C.pp + 0.3 * C.an * C.pn) / C.all**2
        num = (0.7 * C.tp + 0.3 * C.tn) / C.all - pe
        den = 1 - pe
        kappa = num / den

        num2 = C.tp / C.all - (C.ap * C.pp) / C.all**2
        den2 = ap_perf.sqrt(C.ap * C.pp * C.an * C.pn) / C.all**2
        mcc = num2 / den2

        return 0.3 * kappa + 0.7 * mcc

novel_metric = NovelMetric()
novel_metric.initialize()
novel_metric.enforce_special_case_positive()
novel_metric.enforce_special_case_negative()
```

Empirical Results

Datasets:
20 UCI Datasets,
MNIST, Fashion MNIST

Neural Networks:
Multi Layer Perceptron,
Convolutional NN

Performance Metrics:
1) Accuracy
2) F1 score
3) F2 score
4) Geom. Prec. Rec. (GPR)
5) Mathews Cor. Coef. (MCC)
6) Cohen's Kappa score



Summary

	Statistical Consistency	Support Neural Network Learning	Support Custom Metrics	Easy Interface for Practitioners (to optimize custom metrics)
SVM-Perf (Joachim, 2005)	✗	✗	✓	✗
Plug-in based classifiers (Koyejo et al, 2014; Narashiman et al, 2014)	✓	✗	✗	✗
Global objectives (Eban et al, 2014)	✗	✓	✗	✗
DAME & DUPLE (Sanyal et al, 2018)	✗	✓	✗	✗
AP-Perf (Fathony & Kolter, our method)	✓	✓	✓	✓

Download

http://proceedings.mlr.press/v108/fathony20a.html

install:

pip install ap_perf

github:

https://github.com/rizalzaf/ap_perf

install:

]add AdversarialPrediction

github:

https://github.com/rizalzaf/AdversarialPrediction.jl

Thank You