Three faces of binary classification

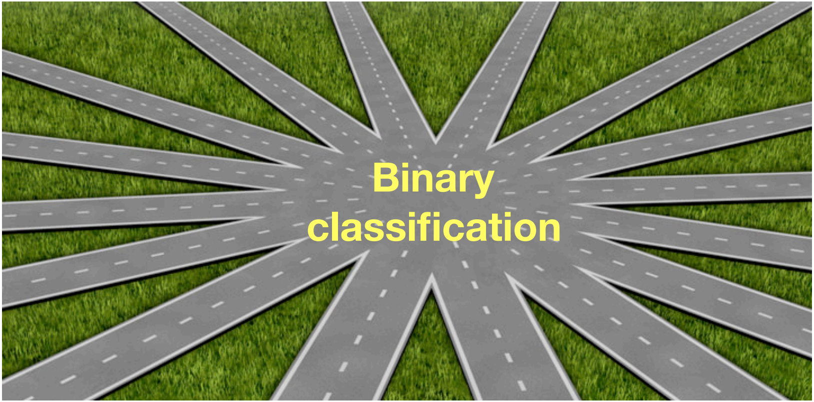
Aditya Krishna Menon

The Australian National University

May 21st, 2018

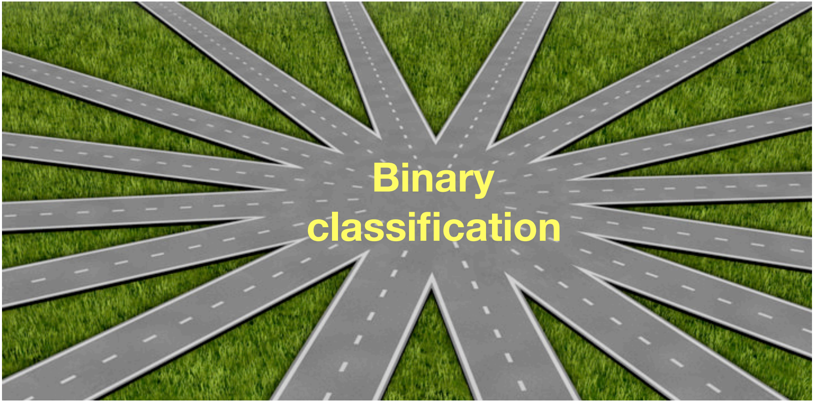
Today's lesson

All roads lead to binary classification



Today’s lesson

All roads lead to binary classification



But what **is** binary classification, exactly?

Recap: binary classification

Goal: predict binary label $y \in \{0, 1\}$ for instance $\mathbf{x} \in \mathcal{X}$

- we call $y = 1$ the "positive" class, and $y = 0$ the "negative" class

Recap: binary classification

Goal: predict binary label $y \in \{0,1\}$ for instance $\mathbf{x} \in \mathcal{X}$

- we call $y = 1$ the "positive" class, and $y = 0$ the "negative" class

We learn a predictive model from a training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$

Recap: binary classification

Goal: predict binary label $y \in \{0, 1\}$ for instance $\mathbf{x} \in \mathcal{X}$

- we call $y = 1$ the "positive" class, and $y = 0$ the "negative" class

We learn a predictive model from a training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$

Canonical models: SVMs, logistic regression

Recap: binary classification

Goal: predict binary label $y \in \{0, 1\}$ for instance $\mathbf{x} \in \mathcal{X}$

- we call $y = 1$ the "positive" class, and $y = 0$ the "negative" class

We learn a predictive model from a training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$

Canonical models: SVMs, logistic regression

Recap: logistic regression

Logistic regression models the probability of an instance $\mathbf{x}$ belonging to the positive class $y = 1$

Recap: logistic regression

Logistic regression models the probability of an instance $\mathbf{x}$ belonging to the positive class $y = 1$

We posit this probability is

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

Recap: logistic regression

Logistic regression models the probability of an instance $\mathbf{x}$ belonging to the positive class $y = 1$

We posit this probability is

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

Classify $\mathbf{x}$ as positive if $\mathbb{P}(y = 1 \mid \mathbf{x}) > 0.5$

Logistic regression is classification?

We informally call logistic regression a “classifier”

Logistic regression is classification?

We informally call logistic regression a “classifier”

But the story is a bit more nuanced:

Logistic regression is classification?

We informally call logistic regression a “classifier”

But the story is a bit more nuanced:

$$\mathbf{1}\left[\frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} > 0.5\right]$$

Logistic regression is classification?

We informally call logistic regression a “classifier”

But the story is a bit more nuanced:

$$\frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} \longrightarrow \mathbf{1}\left[\frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} > 0.5\right]$$

Logistic regression is classification?

We informally call logistic regression a “classifier”

But the story is a bit more nuanced:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} \longrightarrow \frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} \longrightarrow \mathbf{1}\left[\frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} > 0.5\right]$$

Logistic regression is classification?

We informally call logistic regression a "classifier"

But the story is a bit more nuanced:

$$
\begin{array}{ccccc}
\mathbf{w}^{\mathrm{T}}\mathbf{x} & \longrightarrow & \frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} & \longrightarrow & \mathbf{1}\left[\frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}} > 0.5\right] \\
\pitchfork & & \pitchfork & & \pitchfork \\
\mathbb{R} & & [0,1] & & \{0,1\} \\
\text{Scores} & & \text{Probabilities} & & \text{Classifications}
\end{array}
$$

Classifiers, probability estimators, scorers

We may call a model:

$c\colon \mathcal{X} \to \{0,1\}$ a classifier

$p\colon \mathcal{X} \to [0,1]$ a probability estimator

$s\colon \mathcal{X} \to \mathbb{R}$ a scorer

Classifiers, probability estimators, scorers

We may call a model:

$c\colon \mathcal{X} \to \{0,1\}$ a classifier

$p\colon \mathcal{X} \to [0,1]$ a probability estimator

$s\colon \mathcal{X} \to \mathbb{R}$ a scorer

Logistic regression has a scorer

$$\begin{array}{c} s(\mathbf{x}) \\ \cap \\ \mathbb{R} \end{array}$$

Classifiers, probability estimators, scorers

We may call a model:

$c\colon \mathcal{X} \to \{0,1\}$ a classifier

$p\colon \mathcal{X} \to [0,1]$ a probability estimator

$s\colon \mathcal{X} \to \mathbb{R}$ a scorer

Logistic regression has a scorer, which is implicitly converted to a probability estimator

$$
\begin{array}{ccc}
s(\mathbf{x}) & \longrightarrow & \frac{1}{1+e^{-s(\mathbf{x})}} \\
\cap & & \cap \\
\mathbb{R} & & [0,1]
\end{array}
$$

Classifiers, probability estimators, scorers

We may call a model:

$c\colon \mathcal{X} \to \{0,1\}$ a classifier

$p\colon \mathcal{X} \to [0,1]$ a probability estimator

$s\colon \mathcal{X} \to \mathbb{R}$ a scorer

Logistic regression has a scorer, which is implicitly converted to a probability estimator, and then a classifier

$$\underset{\mathbb{R}}{\underset{\cap}{s(\mathbf{x})}} \longrightarrow \underset{[0,1]}{\underset{\cap}{\frac{1}{1+e^{-s(\mathbf{x})}}}} \longrightarrow \underset{\{0,1\}}{\underset{\cap}{\mathbf{1}\left[\frac{1}{1+e^{-s(\mathbf{x})}} > 0.5\right]}}$$

Where are they useful?

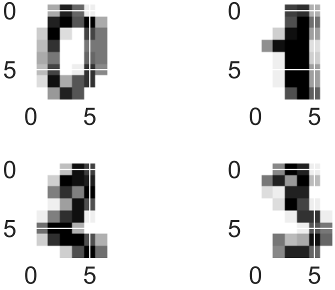
These models provide different things:

Classifiers	hard decisions
Probability-estimators	soft decisions (i.e., confidences)
Scorers	soft-er decisions (i.e., rankings)

Model types: example

Consider predicting if a digit is even or odd

```
(X, Y) = load_digits(return_X_y = True)

for i in range(0, 4):
    plt.subplot(2,2,i+1);
    plt.imshow(X[i,:].reshape((8,8)), cmap = plt.cm.
        gray_r, interpolation='nearest')
```



Model types: example

Consider predicting if a digit is even or odd

```
lrn = LogisticRegression()
lrn.fit(X, (Y % 2 == 0).astype(int))

print(lrn.predict(X[0,:].reshape(1,-1)))
print(lrn.predict_proba(X[0,:].reshape(1,-1))[:,1])
print(lrn.decision_function(X[0,:].reshape(1,-1)))
```

Model types: example

Consider predicting if a digit is even or odd

```
lrn = LogisticRegression()
lrn.fit(X, (Y % 2 == 0).astype(int))

print(lrn.predict(X[0,:].reshape(1,-1)))
print(lrn.predict_proba(X[0,:].reshape(1,-1))[:,1])
print(lrn.decision_function(X[0,:].reshape(1,-1)))
```

gives:

`[1]` (classification)

`[0.99702005]` (probability)

`[5.81286542]` (score)

Are they really different?

Informally, it is understood what one means when calling logistic regression a “classifier”

Are they really different?

Informally, it is understood what one means when calling logistic regression a “classifier”

Care is needed when evaluating the different types of models

Evaluating models

Evaluating models

The general principle for evaluation is:

Our model should discriminate between the two classes

Evaluating models

The general principle for evaluation is:

Our model should discriminate between the two classes

The precise meaning of “discriminate” varies:

Classifiers	have prediction equal to the target label
Probability-estimators	have probability close to the target label
Scorers	score positive instances higher than negative instances

Evaluating models: summary

The general principle for evaluation is:

Our model should discriminate between the two classes

The precise meaning of “discriminate” varies:

Classifiers	misclassification error
Probability-estimators	log-loss
Scorers	pairwise disagreement

Evaluating models: summary

The general principle for evaluation is:

Our model should discriminate between the two classes

The precise meaning of “discriminate” varies:

Classifiers	**misclassification error**
Probability-estimators	log-loss
Scorers	**pairwise disagreement**

Evaluating a classifier

Suppose one trains a classifier $c\colon \mathcal{X} \to \{0, 1\}$

How do we tell if c is “good”, or not?

Evaluating a classifier

Suppose one trains a classifier $c \colon \mathcal{X} \to \{0, 1\}$

How do we tell if c is "good", or not?

Natural thought: look at the misclassification error

$$\mathrm{ERR}(c) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}[y_n \neq c(\mathbf{x}_n)],$$

i.e., the fraction of erroneous classifications

Evaluating a classifier: example

```
(X, Y) = load_digits(return_X_y = True)

...

XTr, XTe, YTr, YTe = train_test_split(X, (Y % 2 == 0).
    astype(int), random_state = 42)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)

1 - accuracy_score(YTe, lrn.predict(XTe))
```

Evaluating a classifier: example

```
(X, Y) = load_digits(return_X_y = True)

...

XTr, XTe, YTr, YTe = train_test_split(X, (Y % 2 == 0).
    astype(int), random_state = 42)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)

1 - accuracy_score(YTe, lrn.predict(XTe))
```

We get a misclassification error of `6.7%`: pretty good!

Evaluating a scorer

Suppose one trains a scorer $s \colon \mathcal{X} \to \mathbb{R}$

How do we tell if s is "good", or not?

Evaluating a scorer

Suppose one trains a scorer $s \colon \mathcal{X} \to \mathbb{R}$

How do we tell if s is "good", or not?

We could look at either:

- how accurate our derived classifier is
- if our scores discriminate the two classes

Evaluating a scorer

Suppose one trains a scorer $s\colon \mathcal{X} \to \mathbb{R}$

How do we tell if s is "good", or not?

We could look at either:

- how accurate our derived classifier is
- if our scores discriminate the two classes

We'll return to the first option later

Evaluating a scorer

Intuitively, s is bad if it scores instances with $y = 0$ higher than those with $y = 1$

Evaluating a scorer

Intuitively, s is bad if it scores instances with $y = 0$ higher than those with $y = 1$

We might measure this using the pairwise-disagreement:

$$\mathrm{PD}(s) = \frac{1}{N_0 \cdot N_1} \sum_{n \colon y_n = 1} \sum_{m \colon y_m = 0} \mathbf{1}[s(\mathbf{x}_n) < s(\mathbf{x}_m)]$$

where $N_i = \#$ instances with $y_n = i$

- fraction of positives scored below negatives

Evaluating a scorer

```
(X, Y) = load_digits(return_X_y = True)

...

XTr, XTe, YTr, YTe = train_test_split(X, (Y % 2 == 0).
    astype(int), random_state = 42)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)

1 - roc_auc_score(YTe, lrn.decision_function(XTe))
```

Evaluating a scorer

```
(X, Y) = load_digits(return_X_y = True)

...

XTr, XTe, YTr, YTe = train_test_split(X, (Y % 2 == 0).
    astype(int), random_state = 42)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)

1 - roc_auc_score(YTe, lrn.decision_function(XTe))
```

We get a pairwise disagreement of `2.6%`: very good!

Evaluating a scorer

We get a different answer if we use pairwise disagreement to evaluate the classifier:

```
(X, Y) = load_digits(return_X_y = True)

...

XTr, XTe, YTr, YTe = train_test_split(X, (Y % 2 == 0).
    astype(int), random_state = 42)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)

1 - roc_auc_score(YTe, lrn.predict(XTe))
```

Evaluating a scorer

We get a different answer if we use pairwise disagreement to evaluate the classifier:

```
(X, Y) = load_digits(return_X_y = True)

...

XTr, XTe, YTr, YTe = train_test_split(X, (Y % 2 == 0).
    astype(int), random_state = 42)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)

1 - roc_auc_score(YTe, lrn.predict(XTe))
```

We get a pairwise disagreement of $6.7\% \neq 2.6\%$!

Evaluating models: summary

The general principle for evaluation is:

> Our model should discriminate between the two classes

The precise meaning of “discriminate” varies:

Classifiers	misclassification error
Probability-estimators	log-loss
Scorers	pairwise disagreement

Roadmap

We'll look at how classification can be useful in:

- predicting rare events
- imputing missing data (by creating features)
- generating images (by creating labels)

Roadmap

We'll look at how classification can be useful in:

- **predicting rare events**
- imputing missing data (by creating features)
- generating images (by creating labels)

Application: imbalanced learning

Putting our skills to the test

Suppose we are approached by a marketing company

Putting our skills to the test

Suppose we are approached by a marketing company

They want to know which people to send promotional fliers to

Putting our skills to the test

Suppose we are approached by a marketing company

They want to know which people to send promotional fliers to

They offer us historical data on people who were sent fliers, and whether or not they responded

Putting our skills to the test

Suppose we are approached by a marketing company

They want to know which people to send promotional fliers to

They offer us historical data on people who were sent fliers, and whether or not they responded

Natural thought: train a classifier!

Putting our skills to the test

```
M = loadmat('kddcup98.mat');
X = M['X']
Y = M['Y'].flatten()
XTr, XTe, YTr, YTe = train_test_split(X, Y, random_state
    = 42, test_size = 0.20)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)
1 - accuracy_score(YTe, lrn.predict(XTe))
```

Putting our skills to the test

```
M = loadmat('kddcup98.mat');
X = M['X']
Y = M['Y'].flatten()
XTr, XTe, YTr, YTe = train_test_split(X, Y, random_state
    = 42, test_size = 0.20)

lrn = LogisticRegression()
lrn.fit(XTr, YTr)
1 - accuracy_score(YTe, lrn.predict(XTe))
```

We get a misclassification error of `5.0%`: very good!

Putting our skills to the test

We confidently present our classifier to the company

Putting our skills to the test

We confidently present our classifier to the company

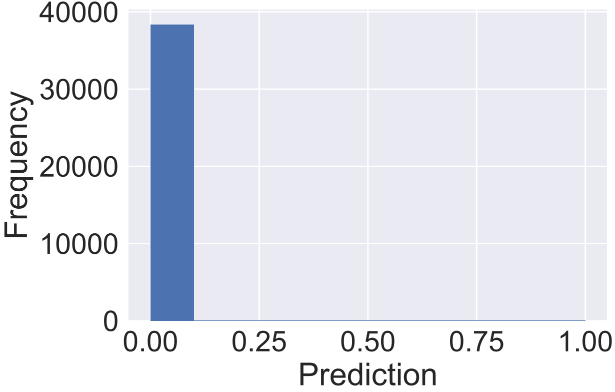
Unfortunately, a week later, they irately fire us

That's all they wrote

When asked why they are unhappy, the company responds:

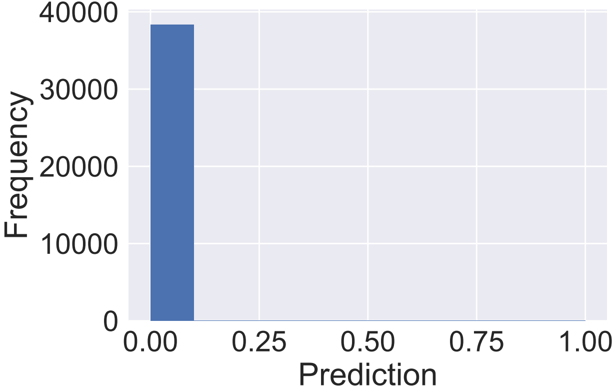
That's all they wrote

When asked why they are unhappy, the company responds:



That's all they wrote

When asked why they are unhappy, the company responds:



We ended up predicting that no one should be sent a flier!

Evaluating a classifier: revisited

Recall that we proposed to compute:

$$\mathrm{ERR}(c) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

Evaluating a classifier: revisited

Recall that we proposed to compute:

$$\mathrm{ERR}(c) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

Suppose that most $y_n = 0$, e.g., most people don't respond

Evaluating a classifier: revisited

Recall that we proposed to compute:

$$\mathrm{ERR}(c) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

Suppose that most $y_n = 0$, e.g., most people don't respond

If we always predicted $c(\mathbf{x}) = 0$, we would find:

$$\mathrm{ERR}(c) = \frac{N_1}{N},$$

where N_1 is the # of instances with $y_n = 1$

Evaluating a classifier: revisited

Recall that we proposed to compute:

$$\mathrm{ERR}(c) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

Suppose that most $y_n = 0$, e.g., most people don't respond

If we always predicted $c(\mathbf{x}) = 0$, we would find:

$$\mathrm{ERR}(c) = \frac{N_1}{N},$$

where N_1 is the # of instances with $y_n = 1$

Since $N_1 \ll N$, the error rate will be very low!

Per-class misclassification error

Intuitively, a trivial classifier is bad because it does not discriminate between the classes

Per-class misclassification error

Intuitively, a trivial classifier is bad because it does not discriminate between the classes

To unwrap this, we could compute the per-class error rates,

$$\mathrm{ERR}_1(c) = \frac{1}{N_1} \sum_{n:\, y_n = 1} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

$$\mathrm{ERR}_0(c) = \frac{1}{N_0} \sum_{n:\, y_n = 0} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

Per-class misclassification error

Intuitively, a trivial classifier is bad because it does not discriminate between the classes

To unwrap this, we could compute the per-class error rates,

$$\mathrm{ERR}_1(c) = \frac{1}{N_1} \sum_{n:\, y_n = 1} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

$$\mathrm{ERR}_0(c) = \frac{1}{N_0} \sum_{n:\, y_n = 0} \mathbf{1}[y_n \neq c(\mathbf{x}_n)]$$

These are known as the false negative and false positive rates

Weighted misclassification error

Standard misclassification error is:

$$\mathrm{ERR}(c) = p \cdot \mathrm{ERR}_1(c) + (1-p) \cdot \mathrm{ERR}_0(c),$$

where $p = \frac{N_1}{N}$ is the fraction of instances with $y_n = 1$

Weighted misclassification error

Standard misclassification error is:

$$\mathrm{ERR}(c) = p \cdot \mathrm{ERR}_1(c) + (1-p) \cdot \mathrm{ERR}_0(c),$$

where $p = \frac{N_1}{N}$ is the fraction of instances with $y_n = 1$

Problem arises because $p \ll 0.5$!

Weighted misclassification error

Standard misclassification error is:

$$\mathrm{ERR}(c) = p \cdot \mathrm{ERR}_1(c) + (1-p) \cdot \mathrm{ERR}_0(c),$$

where $p = \frac{N_1}{N}$ is the fraction of instances with $y_n = 1$

Problem arises because $p \ll 0.5$!

Consider instead a cost-weighted error

$$\mathrm{ERR}(c) = w \cdot \mathrm{ERR}_1(c) + (1-w) \cdot \mathrm{ERR}_0(c),$$

for $w \in [0,1]$ the relative importance of per-class errors

Putting our skills to the test: revisited

```
C = confusion_matrix(YTe, lrn.predict(XTe))
w = 0.5
w * C[0,1] + (1 - w) * C[1,1]
```

Putting our skills to the test: revisited

```
C = confusion_matrix(YTe, lrn.predict(XTe))
w = 0.5
w * C[0,1] + (1 - w) * C[1,1]
```

We get a weighted error rate of `50%`: that sounds very bad!

Putting our skills to the test: revisited

```
C = confusion_matrix(YTe, lrn.predict(XTe))
w = 0.5
w * C[0,1] + (1 - w) * C[1,1]
```

We get a weighted error rate of `50%`: that sounds very bad!

We might now use this measure to compare different classifiers

Putting our skills to the test: revisited

```
C = confusion_matrix(YTe, lrn.predict(XTe))
w = 0.5
w * C[0,1] + (1 - w) * C[1,1]
```

We get a weighted error rate of `50%`: that sounds very bad!

We might now use this measure to compare different classifiers

More abstractly, we are summarising a confusion matrix

Putting our skills to the test

We could also try to evaluate our underlying scorer:

```
M = loadmat('kddcup98.mat');

...

lrn = LogisticRegression()
lrn.fit(XTr, YTr)
1 - roc_auc_score(YTe, lrn.decision_function(XTe))
```

Putting our skills to the test

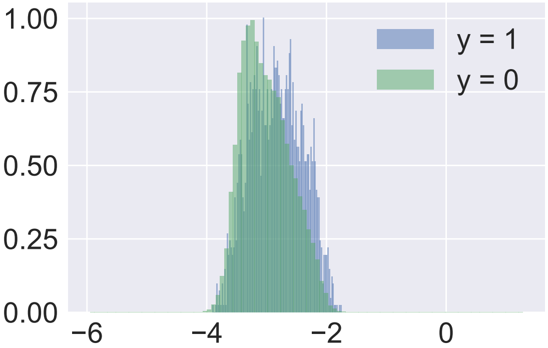
We could also try to evaluate our underlying scorer:

```
M = loadmat('kddcup98.mat');

...

lrn = LogisticRegression()
lrn.fit(XTr, YTr)
1 - roc_auc_score(YTe, lrn.decision_function(XTe))
```

We get a pairwise disagreement of `38.2%`: not great, but not trivial either!

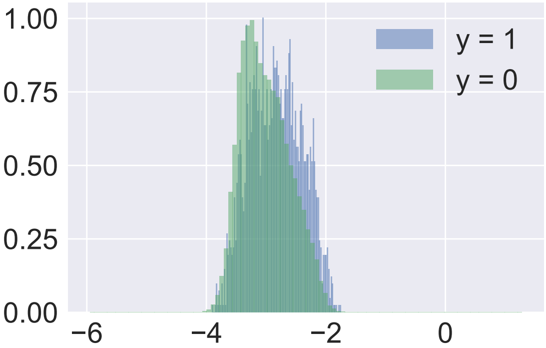
Distribution of scores

There is a slight gap between $y = 1$ and $y = 0$ amongst the scores



Distribution of scores

There is a slight gap between $y = 1$ and $y = 0$ amongst the scores



But also note that all the scores are < 0!

- we are picking a bad threshold to form a classifier!

ROC curves

Given a scorer s, we could make a classifier c_t using any $t \in \mathbb{R}$:

$$c_t(\mathbf{x}) = \mathbf{1}[s(\mathbf{x}) > t]$$

ROC curves

Given a scorer s, we could make a classifier c_t using any $t \in \mathbb{R}$:

$$c_t(\mathbf{x}) = \mathbf{1}[s(\mathbf{x}) > t]$$

The ROC curve is a plot of the resulting false versus true positives, as t is varied:

$$\{(\mathrm{ERR}_0(c_t), 1 - \mathrm{ERR}_1(c_t)) : t \in \mathbb{R}\}$$

ROC curves

Given a scorer s, we could make a classifier c_t using any $t \in \mathbb{R}$:

$$c_t(\mathbf{x}) = \mathbf{1}[s(\mathbf{x}) > t]$$

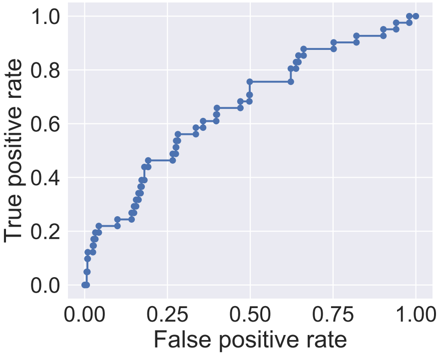
The ROC curve is a plot of the resulting false versus true positives, as t is varied:

$$\{(\mathrm{ERR}_0(c_t), 1 - \mathrm{ERR}_1(c_t)) \colon t \in \mathbb{R}\}$$

This is a graphical summary of all possible classifiers we could obtain by thresholding s

ROC curves: example

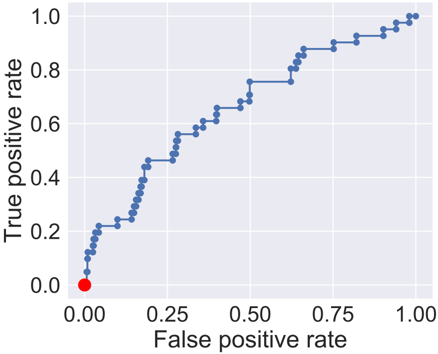
```
fpr, tpr, thresholds = roc_curve(YTe, lrn.
    decision_function(XTe), pos_label = 1)
plt.plot(fpr, tpr, '.-', markersize = 12);
```



Any point on this curve corresponds to a single classifier c_t

ROC curves: example

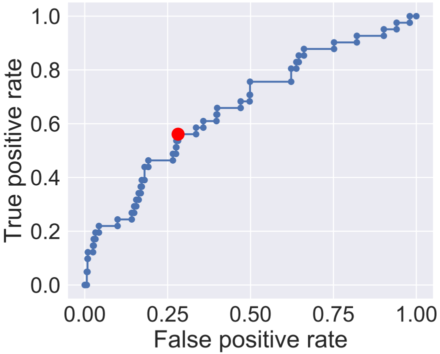
```
fpr, tpr, thresholds = roc_curve(YTe, lrn.
    decision_function(XTe), pos_label = 1)
plt.plot(fpr, tpr, '.-', markersize = 12);
```



Trivial “always negative” classifier: weighted error `50%`

ROC curves: example

```
fpr, tpr, thresholds = roc_curve(YTe, lrn.
    decision_function(XTe), pos_label = 1)
plt.plot(fpr, tpr, '.-', markersize = 12);
```



Better classifier: weighted error `36%`

ROC and pairwise disagreement

It turns out that pairwise disagreement is one minus the area under the ROC curve

ROC and pairwise disagreement

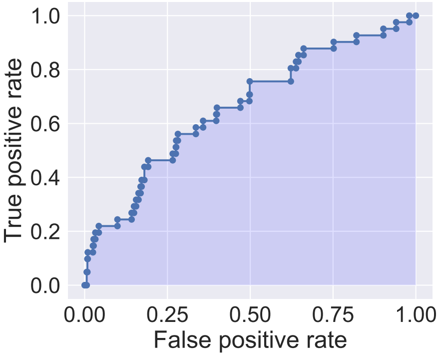
It turns out that pairwise disagreement is one minus the area under the ROC curve

Intuitively, average performance of collection $\{c_t : t \in \mathbb{R}\}$

ROC and pairwise disagreement

It turns out that pairwise disagreement is one minus the area under the ROC curve

Intuitively, average performance of collection $\{c_t : t \in \mathbb{R}\}$



Roadmap

We'll look at how classification can be useful in:

- predicting rare events
- imputing missing data (by creating features)
- generating images (by creating labels)

Roadmap

We'll look at how classification can be useful in:

- predicting rare events
- **imputing missing data (by creating features)**
- generating images (by creating labels)

Application: matrix factorisation

Item response modelling

Suppose an education board approaches us with results from their latest exam

Item response modelling

Suppose an education board approaches us with results from their latest exam

The examiners prepared a number of different questions

Item response modelling

Suppose an education board approaches us with results from their latest exam

The examiners prepared a number of different questions

Each student was give a different subset of these questions

Item response modelling

Suppose an education board approaches us with results from their latest exam

The examiners prepared a number of different questions

Each student was give a different subset of these questions

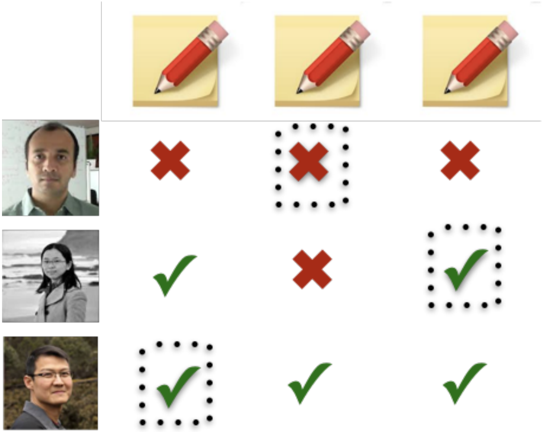
They want to standardise performance across students

Item response modelling: goal

How to account for the fact that some students may have gotten an easy batch of questions?

Item response modelling: strategy

We want to predict how well a student would do on all other questions they weren't asked



Item response modelling: input

Our observed data comprises triplets of the form (student ID, question ID, correct?)

Item response modelling: input

Our observed data comprises triplets of the form (student ID, question ID, correct?)

Compactly, $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, where $\mathbf{x}_n = (s_n, q_n)$ and $y_n \in \{0, 1\}$

Item response modelling: input

Our observed data comprises triplets of the form (student ID, question ID, correct?)

Compactly, $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, where $\mathbf{x}_n = (s_n, q_n)$ and $y_n \in \{0, 1\}$

We want a classifier $c \colon \mathcal{X} \to \{0, 1\}$

- use this to predict unseen (student, question) outcomes

Constructing a classifier

Our inputs $\mathbf{x}_n$ may just be numeric IDs

- e.g., we don't know anything about students apart from their student number
- using this as a raw feature isn't intuitive

Constructing a classifier

Our inputs $\mathbf{x}_n$ may just be numeric IDs

- e.g., we don't know anything about students apart from their student number
- using this as a raw feature isn't intuitive

How can we construct a classifier without any features?!

Constructing a classifier

Our inputs $\mathbf{x}_n$ may just be numeric IDs

- e.g., we don't know anything about students apart from their student number
- using this as a raw feature isn't intuitive

How can we construct a classifier without any features?!

We can try to learn good features from the data!

Constructing a classifier

Construct classifier c via probability-estimator

Constructing a classifier

Construct classifier c via probability-estimator

Recall that logistic regression posits:

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}}$$

Constructing a classifier

Construct classifier c via probability-estimator

Recall that logistic regression posits:

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}}$$

For $\mathbf{x} = (s, q)$, we can posit:

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{u}_s{}^{\mathrm{T}}\mathbf{v}_q}}$$

Constructing a classifier

Construct classifier c via probability-estimator

Recall that logistic regression posits:

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}}$$

For $\mathbf{x} = (s, q)$, we can posit:

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{u}_s{}^{\mathrm{T}}\mathbf{v}_q}}$$

Here, $\mathbf{u}_s$ and $\mathbf{v}_q$ are learned features for the student and question respectively

Training the probability-estimator

For fixed question features $\mathbf{v}_q$,

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-{\mathbf{u}_s}^{\mathrm{T}} \mathbf{v}_q}}$$

is a logistic model with features $\mathbf{v}_q$ and parameters $\mathbf{u}_s$!

Training the probability-estimator

For fixed question features $\mathbf{v}_q$,

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{u}_s{}^{\mathrm{T}}\mathbf{v}_q}}$$

is a logistic model with features $\mathbf{v}_q$ and parameters $\mathbf{u}_s$!

Similarly for fixed student features $\mathbf{u}_s$, we are fitting a logistic model with features $\mathbf{u}_s$ and parameters $\mathbf{v}_q$

Training the probability-estimator

For fixed question features $\mathbf{v}_q$,

$$\mathbb{P}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{u}_s{}^{\mathrm{T}}\mathbf{v}_q}}$$

is a logistic model with features $\mathbf{v}_q$ and parameters $\mathbf{u}_s$!

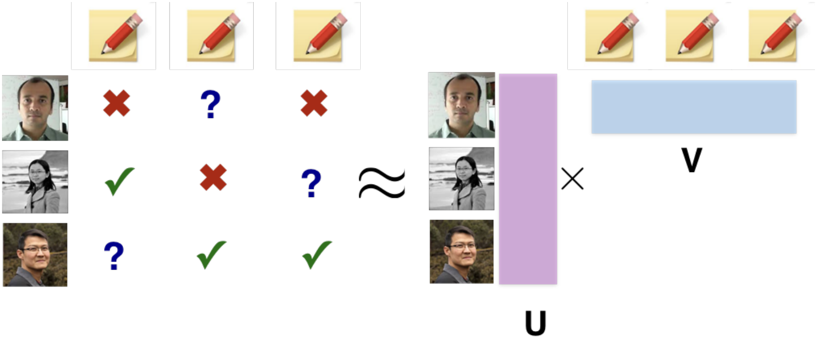
Similarly for fixed student features $\mathbf{u}_s$, we are fitting a logistic model with features $\mathbf{u}_s$ and parameters $\mathbf{v}_q$

We can fit the model by alternating optimisation:

- fix $\{\mathbf{u}_s\}$, and then fit $\{\mathbf{v}_q\}$ via logistic regression
- fix $\{\mathbf{v}_q\}$, and then fit $\{\mathbf{u}_s\}$ via logistic regression
- iterate till convergence

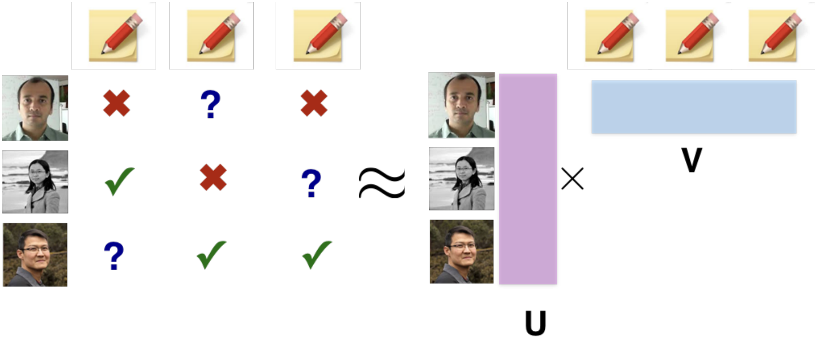
Matrix factorisation view

This can also be understood as a form of nonlinear matrix factorisation (c.f. PCA)



Matrix factorisation view

This can also be understood as a form of nonlinear matrix factorisation (c.f. PCA)



Compared to e.g. PCA, account for missing data

Other applications

Same idea applicable for recommender systems

Roadmap

We'll look at how classification can be useful in:

- predicting rare events
- imputing missing data (by creating features)
- generating images (by creating labels)

Roadmap

We'll look at how classification can be useful in:

- predicting rare events
- imputing missing data (by creating features)
- **generating images (by creating labels)**

Application: GANs

Generative models

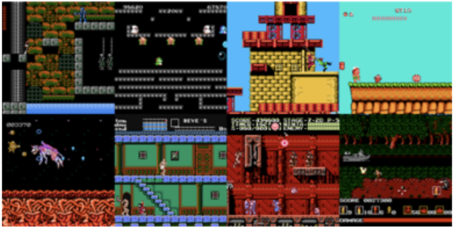
Suppose we want a model that can generate images

From https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

Generative models

Suppose we want a model that can generate images

e.g., from Nintendo game backgrounds

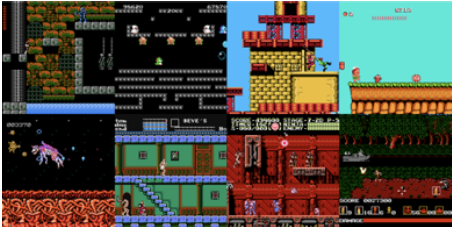


From https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

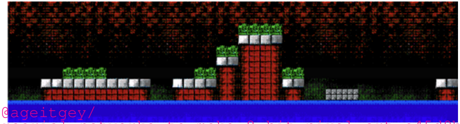
Generative models

Suppose we want a model that can generate images

e.g., from Nintendo game backgrounds



to the background for a new game



From https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

Generative models: formally

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^{N}$, e.g., images

Generative models: formally

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^N$, e.g., images

We want a generator $g \colon \mathcal{Z} \to \mathcal{X}$

Generative models: formally

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^N$, e.g., images

We want a generator $g : \mathcal{Z} \to \mathcal{X}$

We then draw samples $\{g(\mathbf{z}_m)\}_{m=1}^M$, for random seed vectors $\mathbf{z}_m$

Classification problem?

There are no labels available as input

Classification problem?

There are no labels available as input

Hence, we can't possibly treat this as a classification problem

Classification problem?

There are no labels available as input

Hence, we can’t possibly treat this as a classification problem

Unless we create some labels ourselves!

A classification perspective

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^{N}$, e.g., images

A classification perspective

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^N$, e.g., images

Suppose we have generator $g\colon \mathcal{Z} \to \mathcal{X}$, and we draw $\{g(\mathbf{z}_m)\}_{m=1}^M$

A classification perspective

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^N$, e.g., images

Suppose we have generator $g \colon \mathcal{Z} \to \mathcal{X}$, and we draw $\{g(\mathbf{z}_m)\}_{m=1}^M$

How do we tell if g is good, or not?

A classification perspective

We are given a set of instances $\{\mathbf{x}_n\}_{n=1}^N$, e.g., images

Suppose we have generator $g \colon \mathcal{Z} \to \mathcal{X}$, and we draw $\{g(\mathbf{z}_m)\}_{m=1}^M$

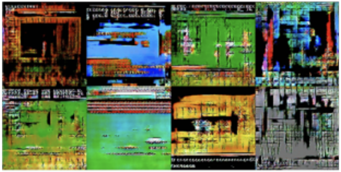
How do we tell if g is good, or not?

Find a classifier to distinguish between $\{\mathbf{x}_n\}_{n=1}^N$ and $\{g(\mathbf{z}_m)\}_{m=1}^M$!

- if a powerful classifier can't tell the difference, then probably humans can't either!

A classification perspective

Generated images



versus

True images

A training objective

Goal: find g whose outputs maximally confuse any classifier!

A training objective

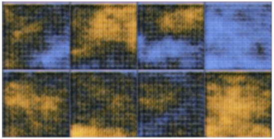
Goal: find g whose outputs maximally confuse any classifier!

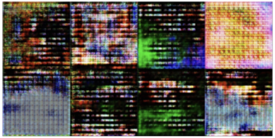
Iterativety optimise generator until its results are indistinguishable from the inputs

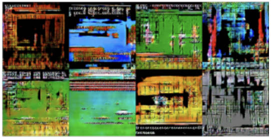
A training objective

Goal: find g whose outputs maximally confuse any classifier!

Iteratively optimise generator until its results are indistinguishable from the inputs

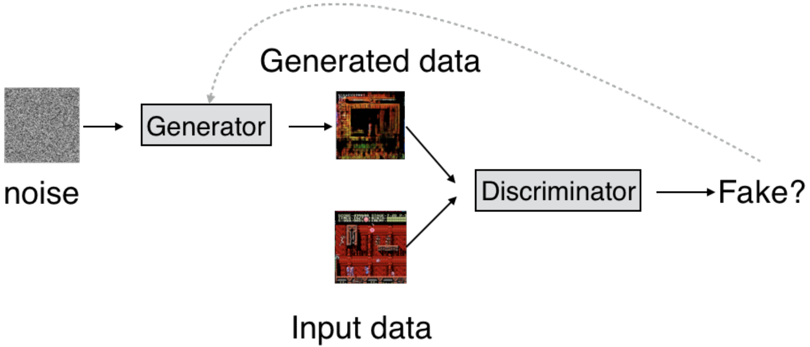






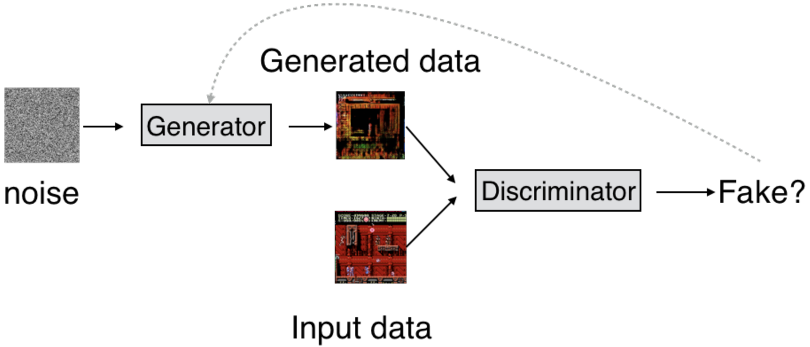
GANs summary

Can think of our procedure as a game between the generator and a discriminator (classifier)



GANs summary

Can think of our procedure as a game between the generator and a discriminator (classifier)



Generative adversarial networks (GANs) implement this idea with neural networks for the generator and discriminator

GAN examples

CAN: Creative Adversarial Networks Generating “Art” by Learning About Styles and Deviating from Style Norms. Elgammal et al., ICCC 2017.

GAN examples

Image-to-Image Translation with Conditional Adversarial Networks. Isola et al., CVPR 2017.

Final thoughts

Summary

Three views of classification

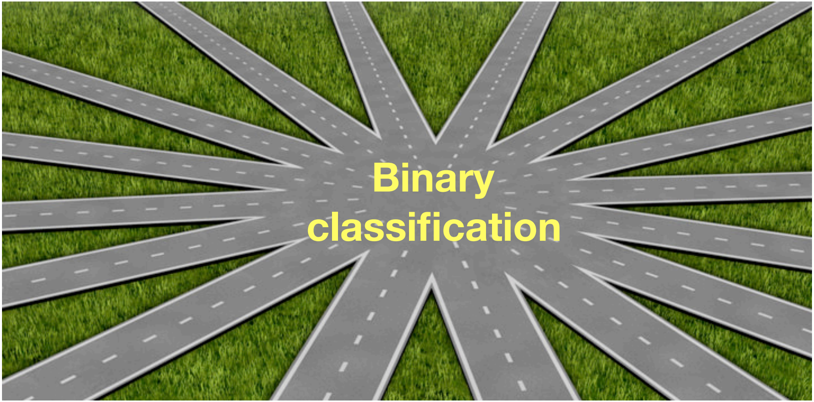
Evaluating classifiers

How classification can be useful in:

- predicting rare events
- imputing missing data (by creating features)
- generating images (by creating labels)

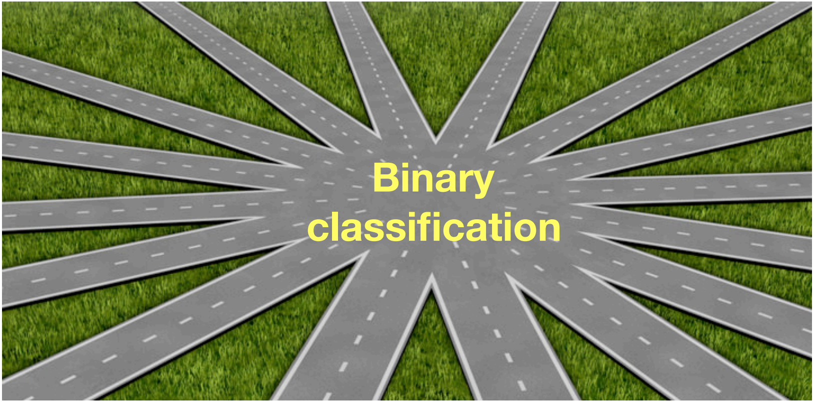
Today's lesson

All roads lead to binary classification



Today's lesson

All roads lead to binary classification



But we need to be careful in defining what "classification" is!