CS 453X: Class 2

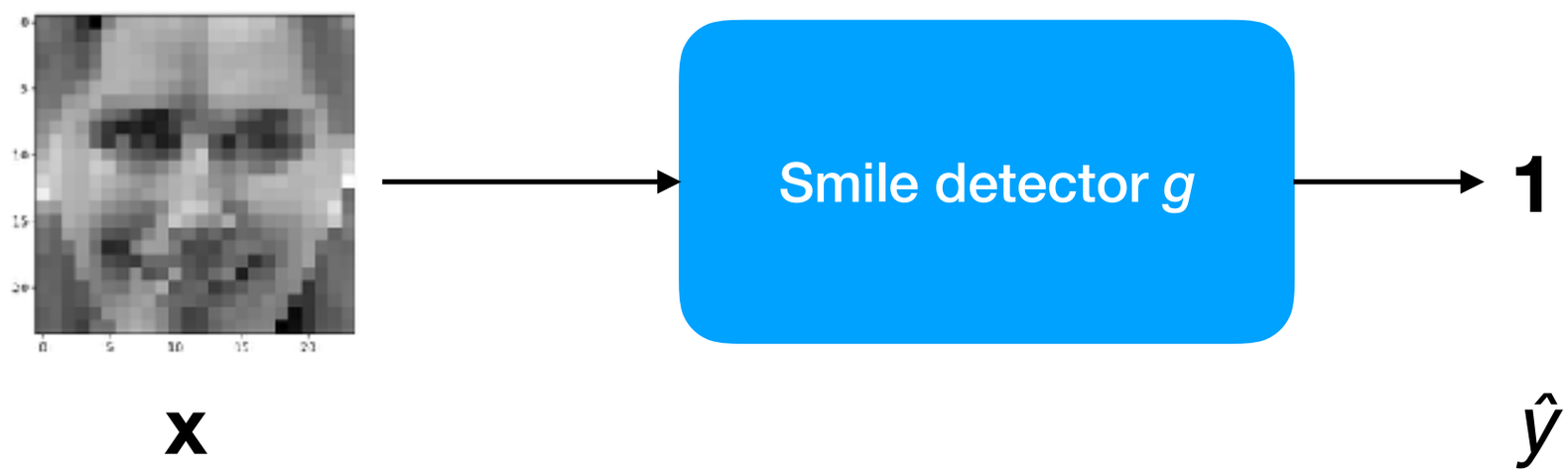
Jacob Whitehill

Automatic smile detection

- Suppose we want to build an automatic smile detector that analyzes a grayscale face image (24x24 pixels) and reports whether the face is smiling.

- We can represent the detector as a function *g* that takes an image **x** as an input and produces a guess $\hat{y}$ as output, where $\mathbf{x} \in \mathbb{R}^{24\times 24}, \hat{y} \in \{0, 1\}$.

- Abstractly, *g* can be considered a “machine”:

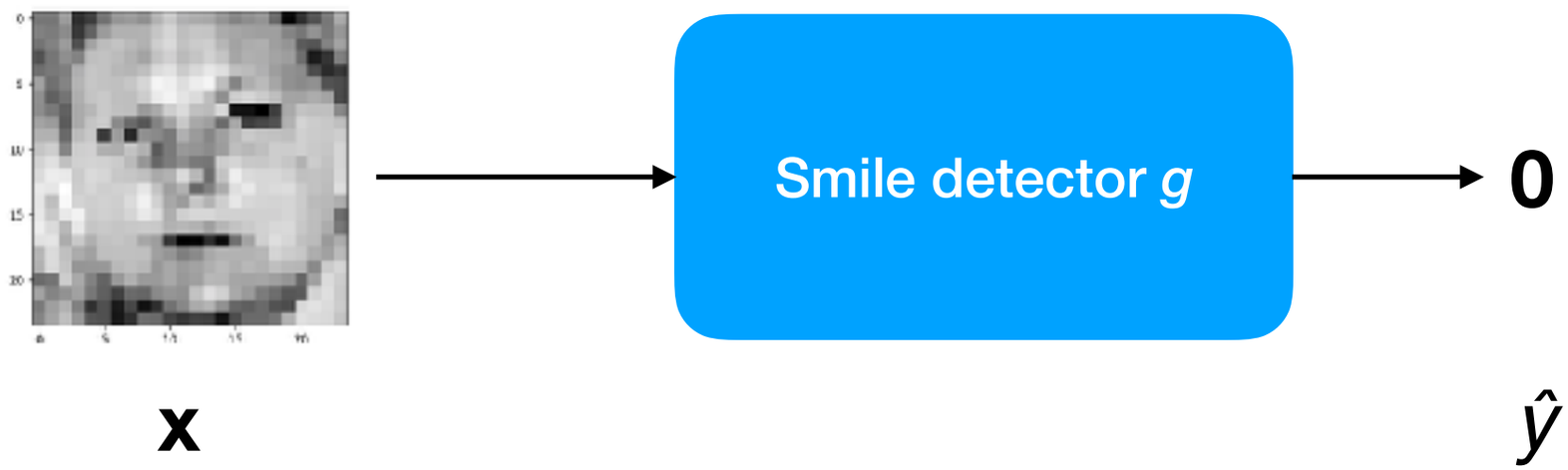


Automatic smile detection

- Suppose we want to build an automatic smile detector that analyzes a grayscale face image (24x24 pixels) and reports whether the face is smiling.

- We can represent the detector as a function *g* that takes an image **x** as an input and produces a guess $\hat{y}$ as output, where $\mathbf{x} \in \mathbb{R}^{24\times 24}, \hat{y} \in \{0, 1\}$.

- Abstractly, *g* can be considered a “machine”:



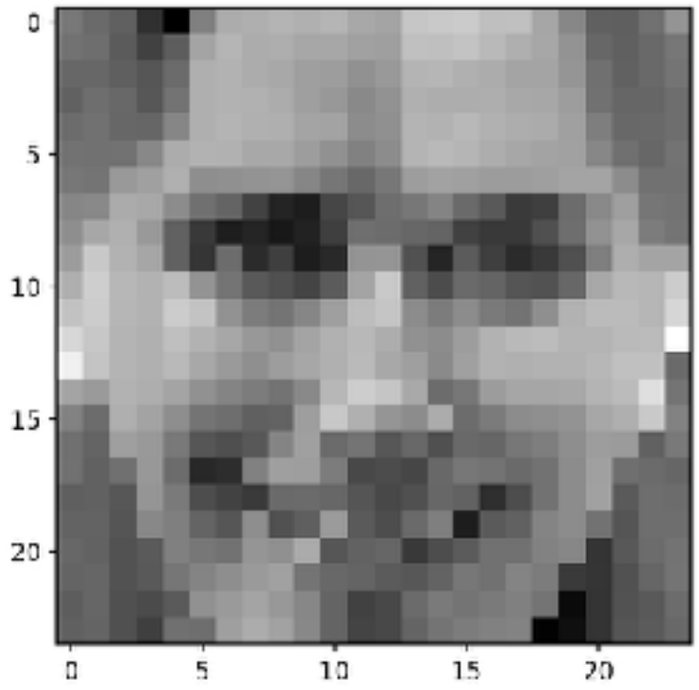
Automatic smile detection

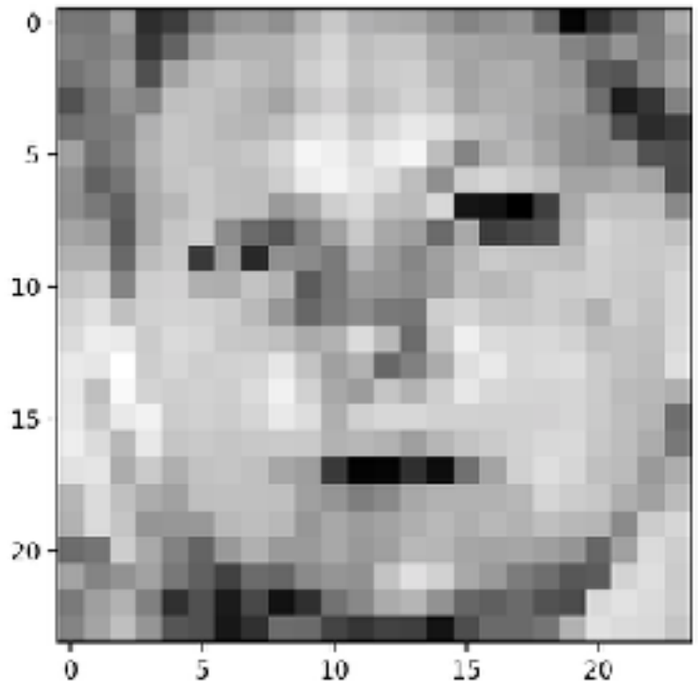
- Suppose we build *g* so that its output depends on only a *single* pair of pixels within the input face:

$$g(\mathbf{x}) = \mathbb{I}[\mathbf{x}_{r_1,c_1} > \mathbf{x}_{r_2,c_2}]$$

- Which pairs (r_1, c_1), (r_2, c_2) would you choose?

- How good is it?





Accuracy measurement

- To evaluate the simple “smile detector” *g*, we need a **test set** $\mathcal{D}^{\text{test}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ consisting of:
  - Images { $\mathbf{x}_i$ } of both classes (smile/1, non-smile/0)
  - Corresponding labels { y_i }.
- The exact composition (ratio of positive/negative examples) is flexible but may impact the way we interpret the machine’s accuracy.

Accuracy measurement

- Let’s try a few examples by hand in smile_demo.py
- What accuracy did we achieve with a single predictor?
- Is this “good”?

Selecting a baseline

- What fraction of faces in $\mathcal{D}^{\text{test}}$ are smiling faces? 54.6%
- How accurate (f_{PC}) would a predictor be that just always output 1 no matter what the image looked like?
  - 54.6%
- Note that there are other accuracy functions (e.g., f_{AUC}) that are invariant to the proportion of each class — aka the **prior probabilities** — of each class in the test set.

Combining multiple predictors

- Determining smile/non-smile based on a single comparison is very weak.
- What if we combined *multiple* pairs and took the majority-vote (choose non-smile if tied) across all *m* comparisons?

$$
\begin{aligned}
g^{(j)}(\mathbf{x}) &= \mathbb{I}[\mathbf{x}_{r_1,c_1} > \mathbf{x}_{r_2,c_2}] \\
\hat{y} = g(\mathbf{x}) &= \mathbb{I}\left[\left(\frac{1}{m}\sum_{j=1}^{m} g^{(j)}(\mathbf{x})\right) > 0.5\right]
\end{aligned}
$$

Combining multiple predictors

- The accuracy of the “ensemble” can vary hugely depending on how the *m* “weak” predictors were selected.

- If the *m* weak predictors tend to give the same answer for the same inputs — i.e., they are **correlated** — then the ensemble predictor may not be much better than any of the weak predictors.

- It is important to choose the *m* weak predictors to work well in *cooperation*.

Combining multiple predictors

- Let's change notation slightly:

$$g^{(j)}(\mathbf{x}) \quad = \quad \mathbb{I}[\phi^{(j)}(\mathbf{x}) > 0]$$

$$\phi^{(j)}(\mathbf{x}) \quad = \quad \mathbf{x}_{r_1,c_1} - \mathbf{x}_{r_2,c_2}$$

- Each $\phi^{(j)}$ is called a **feature** of the input **x**.

- In machine learning*, the features serve as the basis of the machine's predictions.

* With more recent "deep learning" algorithms, the distinction between "feature extraction" and "prediction" is blurred.

Feature set

- Since each $g^{(j)}$ examines only a single feature, choosing a predictor $g^{(j)}$ is equivalent to choosing a feature $\phi^{(j)}$.

- Let the set of all possible features be called $\mathcal{F}$.

- Note that each prediction $\hat{y}$ implicitly depends on $\phi^{(1)}, \ldots, \phi^{(m)}$.

$$\hat{y} = g(\mathbf{x}) = \mathbb{I}\left[\left(\frac{1}{m}\sum_{j=1}^{m} g^{(j)}(\mathbf{x})\right) > 0.5\right]$$

- Our goal is to find the **best** combination of m features, i.e., the one whose accuracy is:

$$\max_{(\phi^{(1)},\ldots,\phi^{(m)})\in\mathcal{F}^m} f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}})$$

Feature set

- What is the size of the feature set:

$$\mathcal{F} = \{(r_1, c_1, r_2, c_2) \in \{0, \ldots, 23\}^4 : (r_1, c_1) \neq (r_2, c_2)\}$$

1. 317952
2. 331200
3. 304704
4. 255024

Feature set

- What is the size of the feature set:

$$\mathcal{F} = \{(r_1, c_1, r_2, c_2) \in \{0, \ldots, 23\}^4 : (r_1, c_1) \neq (r_2, c_2)\}$$

1. 317952
2. 331200
3. 304704
4. 255024 = 24*23*22*21

Feature set

- What is the size of the feature set:

$$\mathcal{F} = \{(r_1, c_1, r_2, c_2) \in \{0, \ldots, 23\}^4 : (r_1, c_1) \neq (r_2, c_2)\}$$

1. 317952
2. 331200
3. 304704 = (24*23)*(24*23)
4. 255024 = 24*23*22*21

Feature set

- What is the size of the feature set:

$$\mathcal{F} = \{(r_1, c_1, r_2, c_2) \in \{0, \ldots, 23\}^4 : (r_1, c_1) \neq (r_2, c_2)\}$$

1. 317952 = (24*23)*(24*24)
2. 331200
3. 304704 = (24*23)*(24*23)
4. 255024 = 24*23*22*21

Feature set

- What is the size of the feature set:

$$\mathcal{F} = \{(r_1, c_1, r_2, c_2) \in \{0, \ldots, 23\}^4 : (r_1, c_1) \neq (r_2, c_2)\}$$

1. 317952 = (24*23)*(24*24)
2. **331200 = (24*24)*(24*24-1)**
3. 304704 = (24*23)*(24*23)
4. 255024 = 24*23*22*21

Feature set

- If $|\mathcal{F}| = 331200$, then even for *m*=5, we have

  $|\mathcal{F}^5|$ = 39852139380159283200000000000

- It is computationally intractable to enumerate over all of these combinations of features!

- Overcoming the exponential computational costs of **brute-force** (“try everything”) optimization is one fo the chief goals of ML research.

Step-wise classification

- Step-wise regression/classification is a **greedy** algorithm for selecting features/predictors **myopically**, i.e., based on “what looks best right now”.

- Instead of optimizing *jointly* to find:

$$\max_{(\phi^{(1)},\ldots,\phi^{(m)})\in\mathcal{F}^m} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\ldots,\phi^{(m)})$$

We sometimes write the parameters that a function depends on after the ;

Step-wise classification

- Step-wise regression/classification is a **greedy** algorithm for selecting features/predictors **myopically**, i.e., based on “what looks best right now”.

- Instead of optimizing *jointly* to find:

$$\max_{(\phi^{(1)},\ldots,\phi^{(m)})\in\mathcal{F}^m} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\ldots,\phi^{(m)})$$

…we optimize *iteratively*:

$$\max_{\phi^{(1)}\in\mathcal{F}} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)})$$

Find the single best feature.

Step-wise classification

- Step-wise regression/classification is a **greedy** algorithm for selecting features/predictors **myopically**, i.e., based on “what looks best right now”.

- Instead of optimizing *jointly* to find:

$$\max_{(\phi^{(1)},\ldots,\phi^{(m)})\in\mathcal{F}^m} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\ldots,\phi^{(m)})$$

…we optimize *iteratively*:

$$\max_{\phi^{(1)}\in\mathcal{F}} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)})$$

$$\max_{\phi^{(2)}\in\mathcal{F}} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\phi^{(2)})$$

Given we have already committed to the first feature, which single next feature is best in combination?

Step-wise classification

- Step-wise regression/classification is a **greedy** algorithm for selecting features/predictors **myopically**, i.e., based on “what looks best right now”.

- Instead of optimizing *jointly* to find:

$$\max_{(\phi^{(1)},\ldots,\phi^{(m)})\in\mathcal{F}^m} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\ldots,\phi^{(m)})$$

…we optimize *iteratively*:

$$\max_{\phi^{(1)}\in\mathcal{F}} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)})$$

$$\max_{\phi^{(2)}\in\mathcal{F}} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\phi^{(2)})$$

$$\max_{\phi^{(3)}\in\mathcal{F}} f_{\mathrm{PC}}(\mathbf{y},\hat{\mathbf{y}};\phi^{(1)},\phi^{(2)},\phi^{(3)})$$ **Repeat.**

...

Step-wise classification

- Instead of $|\mathcal{F}|^m$ possible choices, we only have $m \times |\mathcal{F}|$.
- This is doable!
- We have reduced the exponential growth into linear growth — big difference!
- Note, however, that there is no guarantee that the solution is optimal. Step-wise classification is an **approximate solution** to selecting the *m* best features/predictors.

$$\max_{\phi^{(1)} \in \mathcal{F}} f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}; \phi^{(1)})$$

$$\max_{\phi^{(2)} \in \mathcal{F}} f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}; \phi^{(1)}, \phi^{(2)})$$

$$\max_{\phi^{(3)} \in \mathcal{F}} f_{\mathrm{PC}}(\mathbf{y}, \hat{\mathbf{y}}; \phi^{(1)}, \phi^{(2)}, \phi^{(3)})$$

...

Step-wise classification

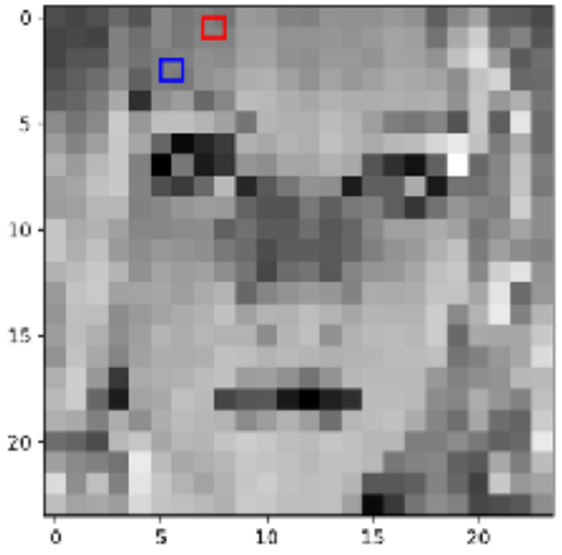
- Pseudocode:

```
predictors = []  # Empty list
For j = 1, ..., m:
  1. Find next best predictor given what's already in predictors
  2. Add it to predictors
```

- Run smile_demo.py and optimize on 10 images.

Step-wise classification

- Accuracy (on 10 images): 100%.
- Learned feature (somewhat counterintuitive):



- What happened?

Overfitting

- When we optimized the *m*=1 features on a set of just 10 images, we discovered a **spurious relationship** between the image **x** and the target label *y*.
    - Spurious: the relationship would not **generalize** to a much larger set of images.
- **Problem**: we have many features (331200) but very few images (10) we need to classify.
    - Out of 331200, it's not hard to find a few features that happen to discriminate smiles/non-smiles just by chance.
- This is called **overfitting** to the dataset.

Training versus testing

- In machine learning, we always optimize the parameters/ features of our classifier/regressor on a **training set** $\mathcal{D}^{\text{train}}$.

- We then measure accuracy on a **testing set** $\mathcal{D}^{\text{test}}$ that is disjoint from (contains no common elements with) the training set.

- The training and testing sets should be collected in the same manner.

  - What does this mean?

Data distribution

- When we collect a dataset (for training or testing), we are **sampling** from some universe of data.
    - Face images from Google Image Search.
    - DNA sequences from patients with a certain disease.
    - Prices of stocks from NYSE.
    - Pages of text from recently published books.
    - …

Data distribution

- Each input **x** is sampled from a probability distribution $p(\mathbf{x})$.
- Each label y is sampled from a conditional probability distribution $p(y \mid \mathbf{x})$.
- Both probability distributions depend on the **universe** of data under consideration.
- How we characterize the accuracy of our trained "machine" will depend crucially on $p(\mathbf{x})$, $p(y \mid \mathbf{x})$.

Data distribution

- Example:
  - Our “universe” is the 2018 population of undergraduate WPI students.
  - We randomly sample a profile picture **x** from the universe according to $p(\mathbf{x})$. This means that:

is less likely than

x **x'**

i.e., $p(\mathbf{x}) < p(\mathbf{x}')$.

Data distribution

- Example:

  - Suppose we're trying to predict each person's favorite weekend activity *y* from their profile picture.

  - WPI students may do different things compared to students at University of Florida (different climate, etc.).

  - The conditional distribution $p(y \mid \mathbf{x})$ — what is the favorite activity *y* does it look like the person has, given what their profile picture **x** looks like — can depend highly on the universe under consideration.

Training versus testing

- The distributions $p(\mathbf{x})$, $p(y \mid \mathbf{x})$ used to sample training data should be the same as the distributions used to sample testing data.

- In practice, we typically sample a large dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and then partition it into training and testing sets.

  - 50%/50% and 80%/20% are common choices.

Training versus testing

- During training, we can do whatever we want to $\mathcal{D}^{\text{train}}$.
- During training, we should **never** look at $\mathcal{D}^{\text{test}}$.
- After training, we compute accuracy on $\mathcal{D}^{\text{test}}$ and report it.
- If we want to change the classifier design, we should re-train and report accuracy on a different test set!

Training versus testing

- This ideal is hard to achieve in practice because labeled data are often difficult to collect.
    - There are some exceptions: computer games.
- In practice:
    - The computer should never examine the test set.
    - We humans should try to minimize how often we examine performance on the test set.

Training versus testing

- Machine learning uses a different approach (training+testing) compared to classical statistics.

- Approach in statistics:
  - Hypothesis tests take into account the **degrees of freedom** (dof), which depends on:
    - Number of tunable parameters
    - Number of examples in the dataset

- Advantage: just one dataset is required.

- Disadvantage: requires strong assumptions on distribution of data and prediction errors.

Training versus testing

- In homework 1 (part II), the training and testing sets are given to you.

- Run smile_demo.py

Weakness of our feature set

- So far, the feature we have considered are very weak:
  - Is pixel (r_1,c_1) brighter than pixel (r_2,c_2)?
- We can't even express simple relationships such as:
  - "(r_1,c_1) is at least 5 bigger than (r_2,c_2)"
  - "2 times (r_1,c_1) is bigger than (r_2,c_2)"
  - "2 times (r_1,c_1) plus 4 times (r_2,c_2) is larger than (r_3,c_3)".

Linear regression

- We can harness these more complex relationships using **linear regression**.

- Let’s switch back to the age estimation problem…

Linear algebra

- A **column vector** is a (*n* x 1) matrix.
- A **row vector** is a (1 x *n*) matrix.
- The **transpose** of (*n* x *k*) matrix **A**, denoted $\mathbf{A}^T$, is (*k* x *n*).
- Multiplication of matrices **A** and **B**:
  - Only possible when: **A** is (*n* x *k*) and **B** is (*k* x *m*)
  - Result: (*n* x *m*)
- The **inner product** between two column vectors (same length) **x**, **y** can be written as: $\mathbf{x}^T\mathbf{y}$