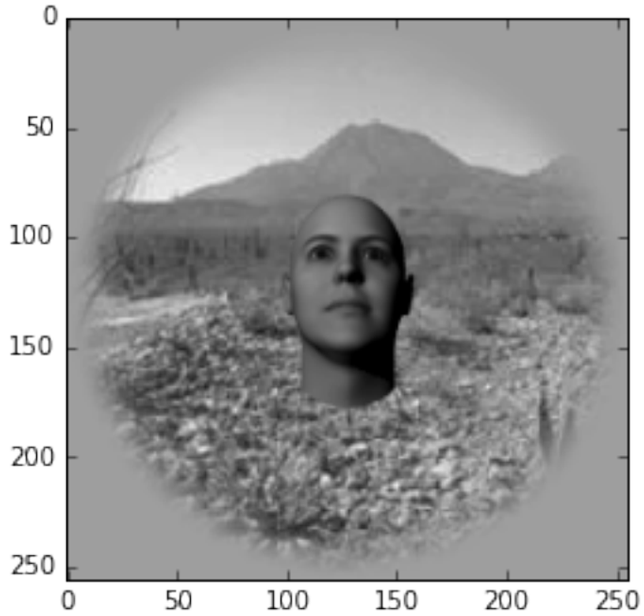
Psych 253
Advanced Statistical Modeling

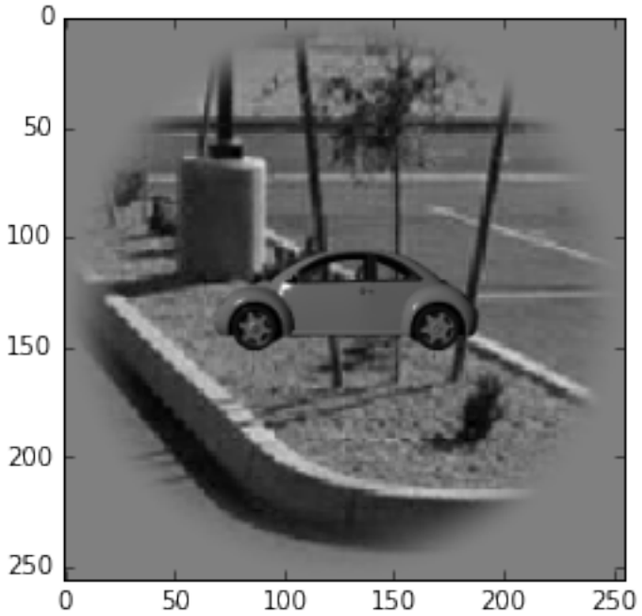
Classification Part 1: Minimum Distance Classifiers

Daniel Yamins
Wu Tsai Neurosciences Institute
Departments of Psychology and Computer Science
Stanford Artificial Intelligence Laboratory
Stanford University

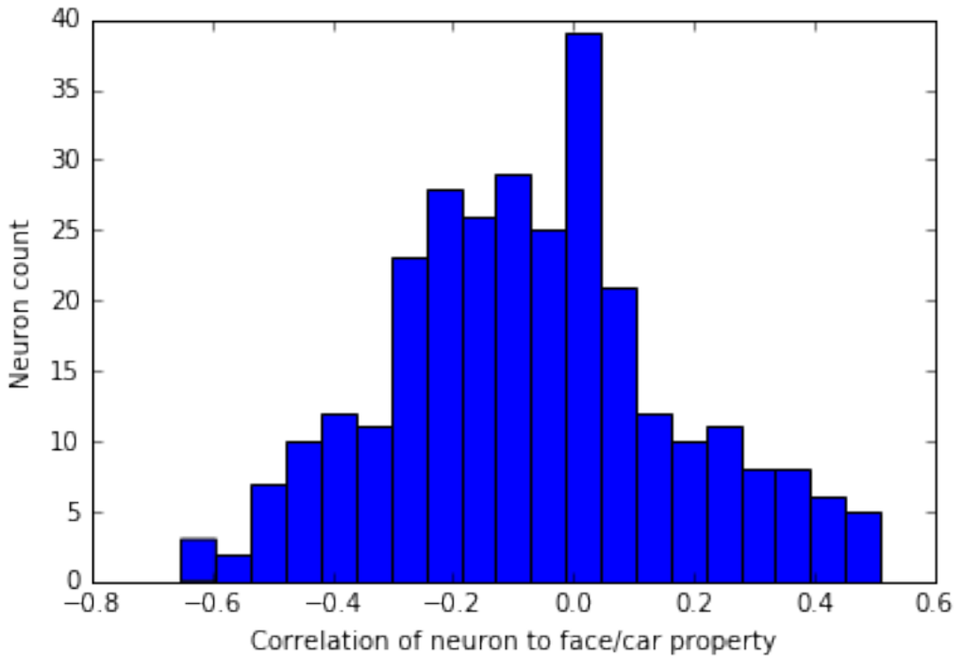
Russ Poldrack
Department of Psychology
Stanford University

Simple correlations of neurons to stimulus properties

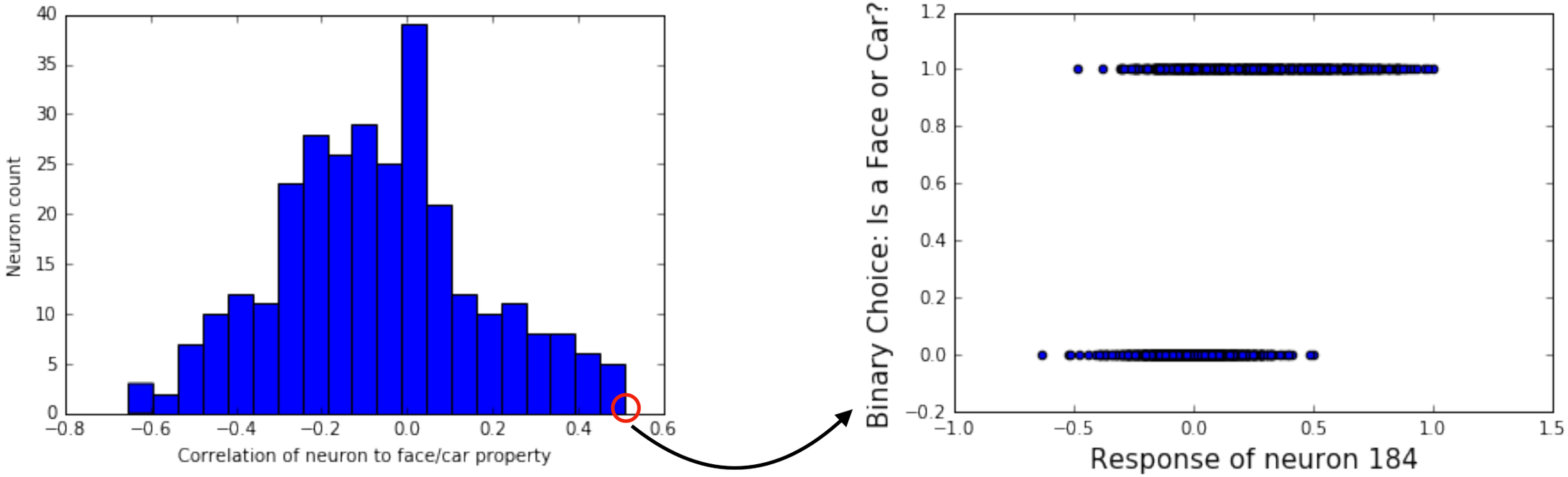




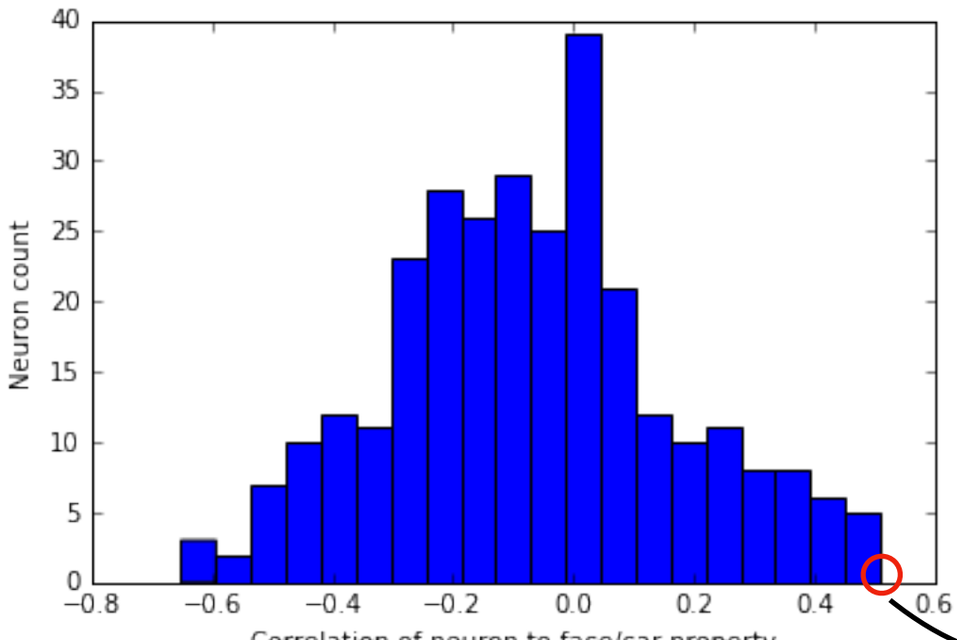
Simple correlations of neurons to stimulus properties

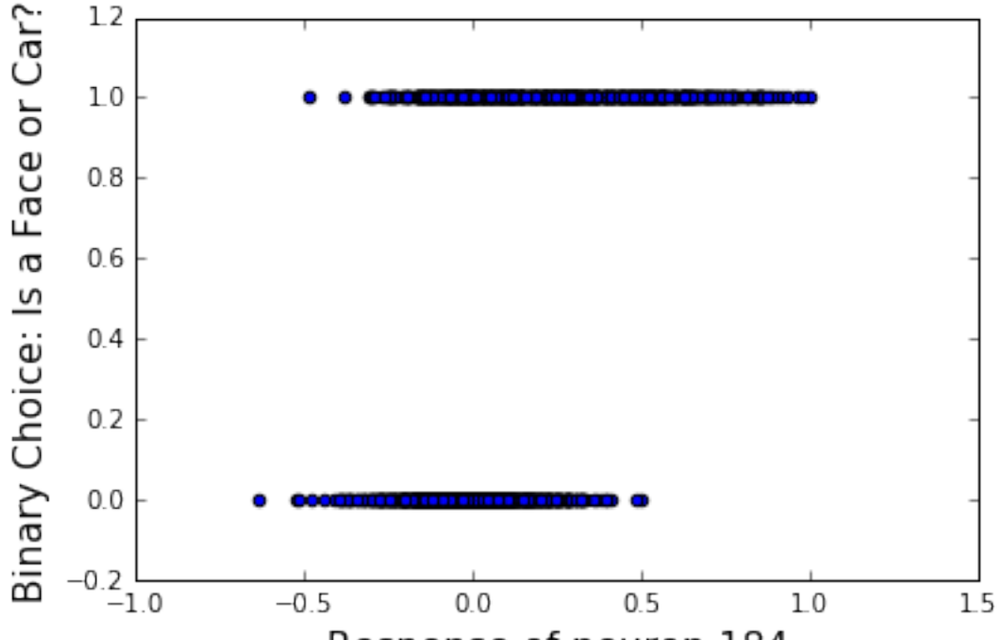


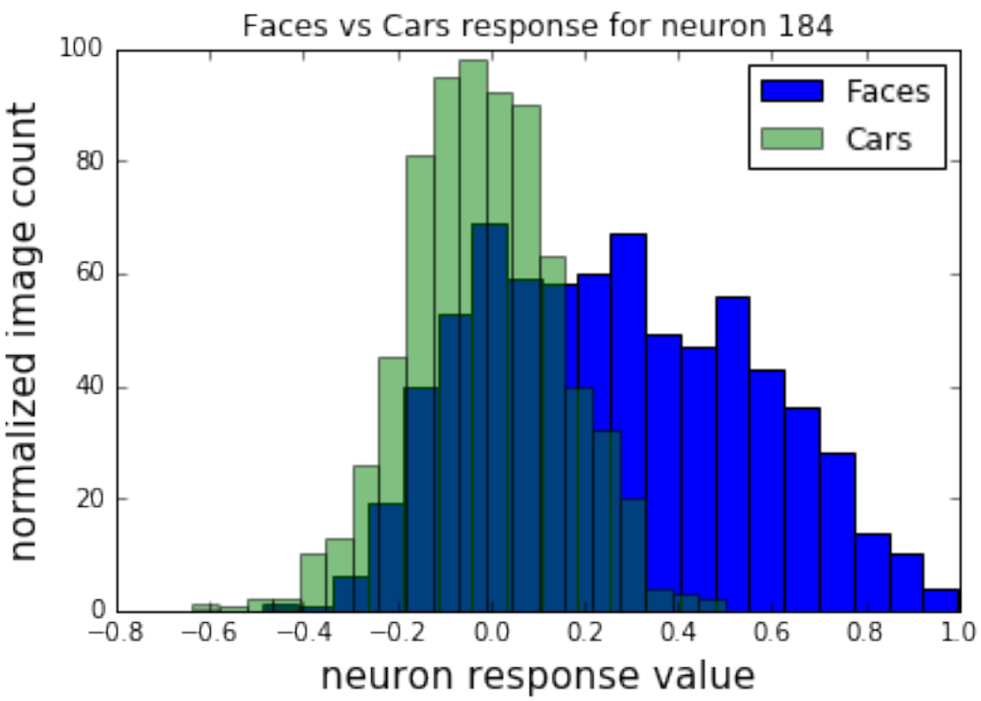
Simple correlations of neurons to stimulus properties



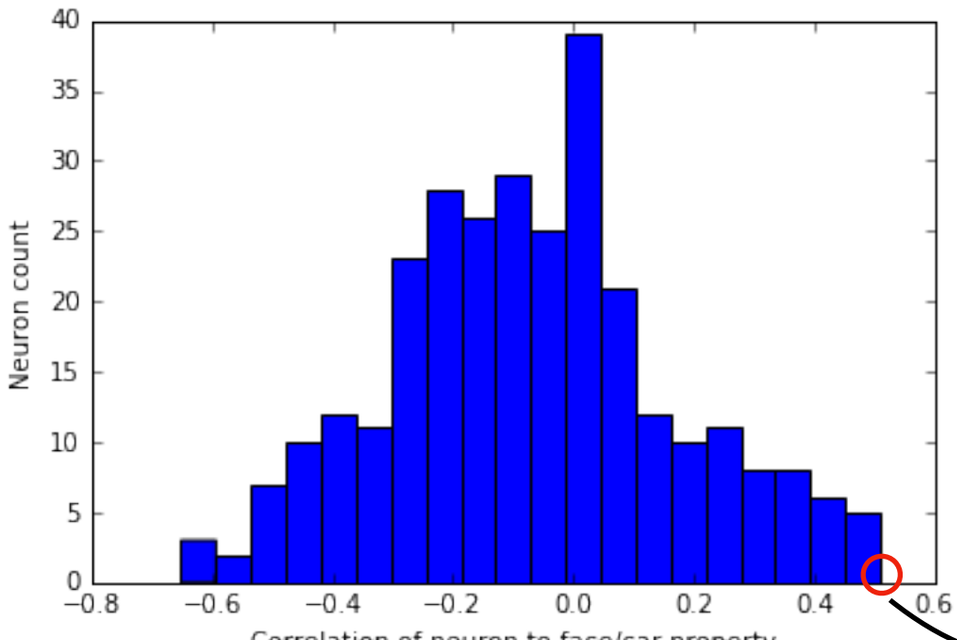
Simple correlations of neurons to stimulus properties

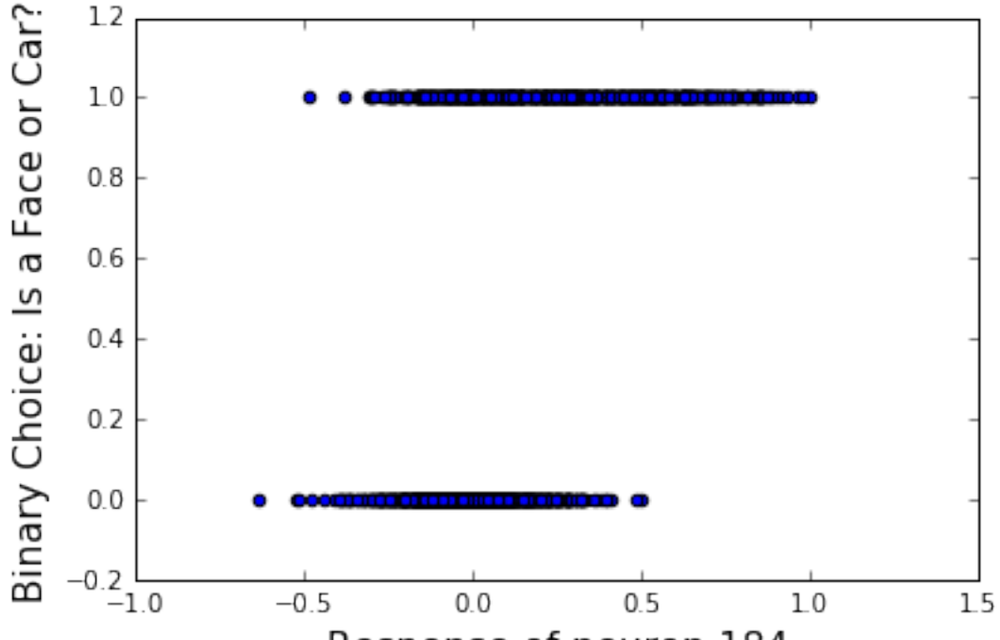


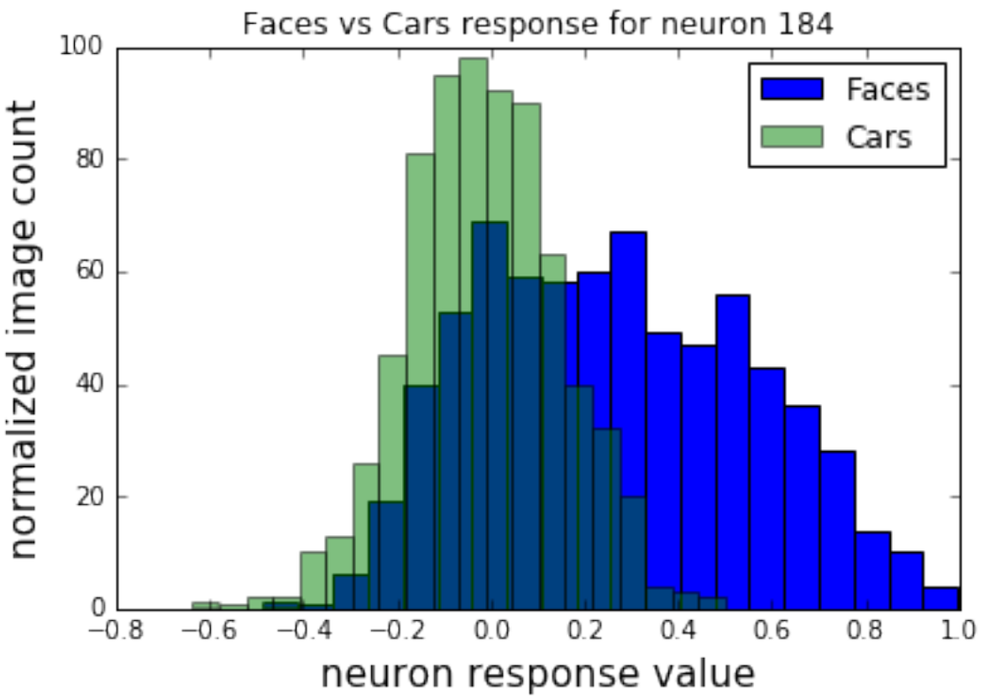


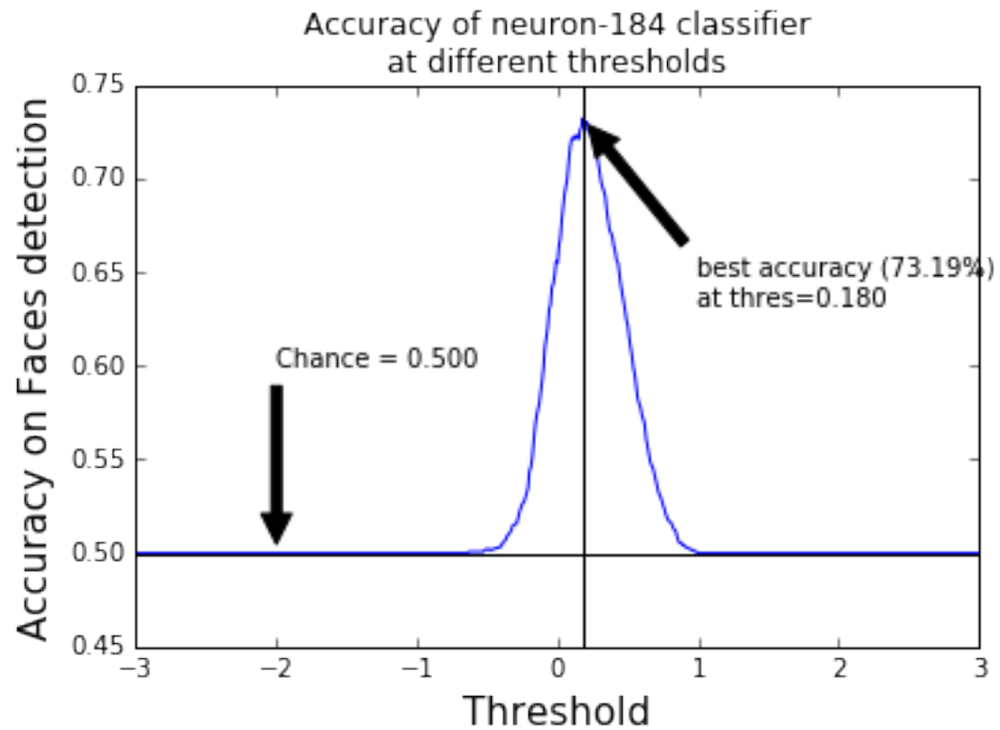


Simple correlations of neurons to stimulus properties

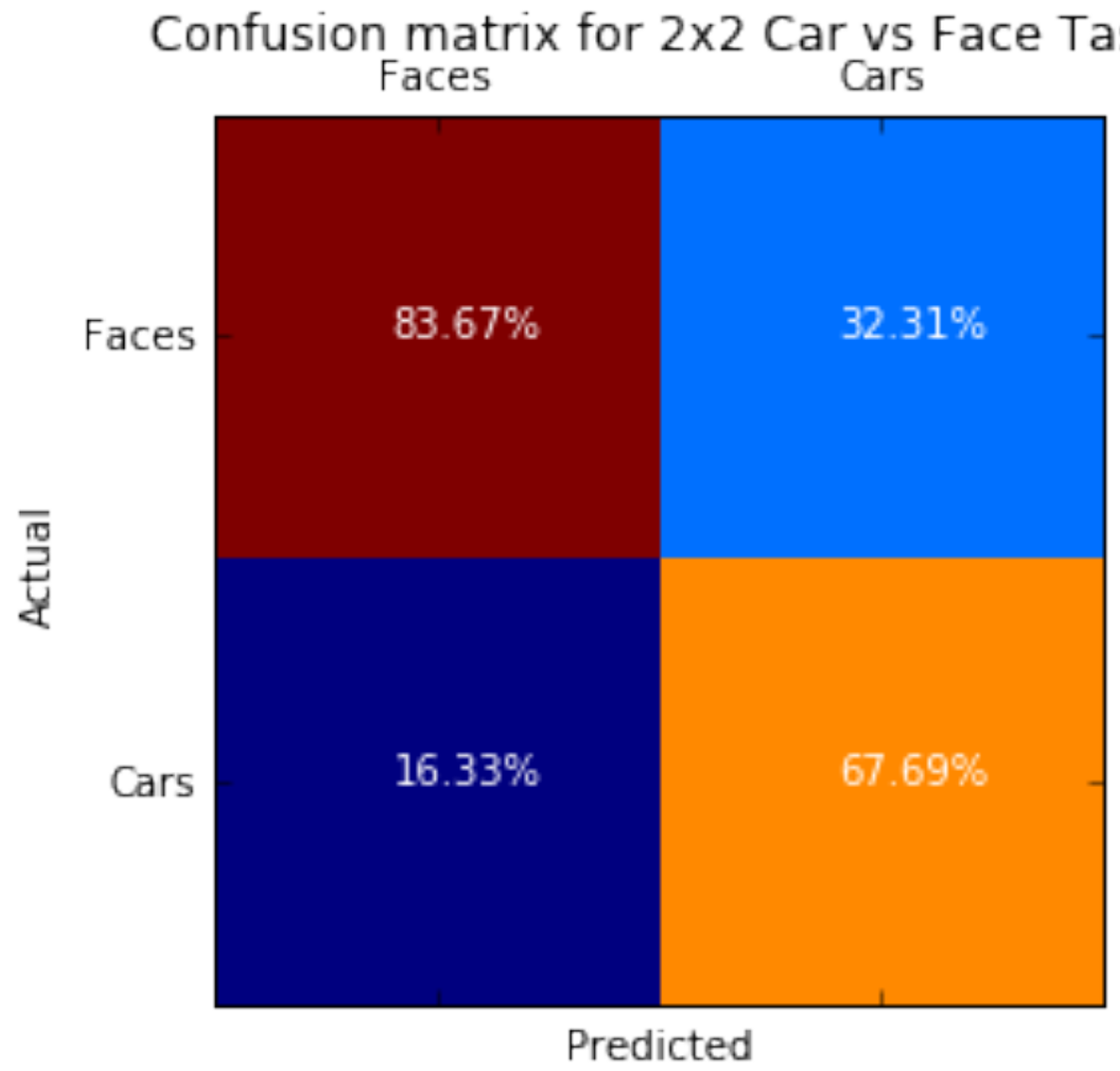






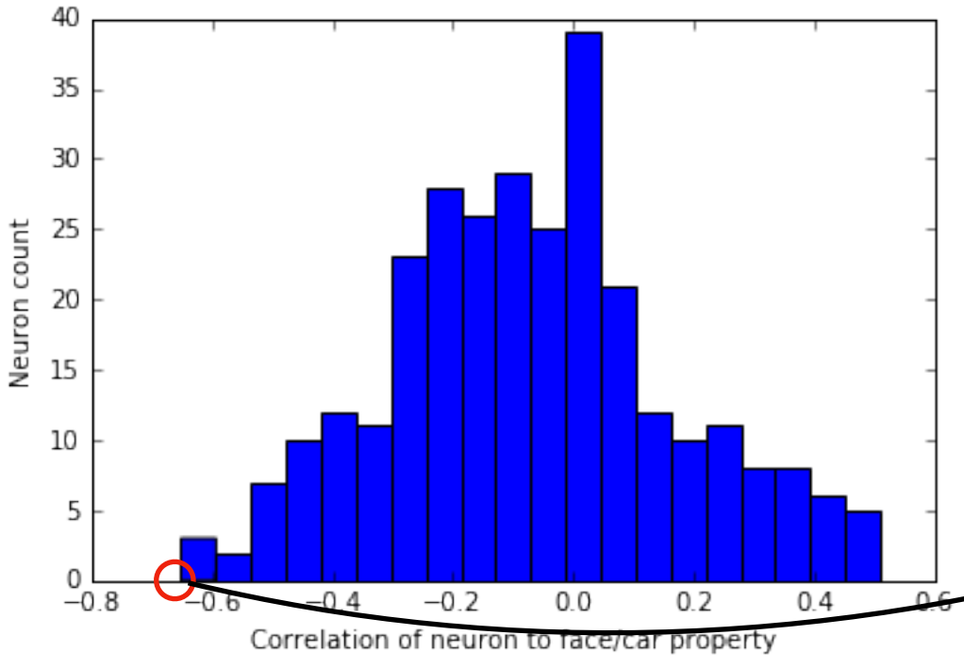


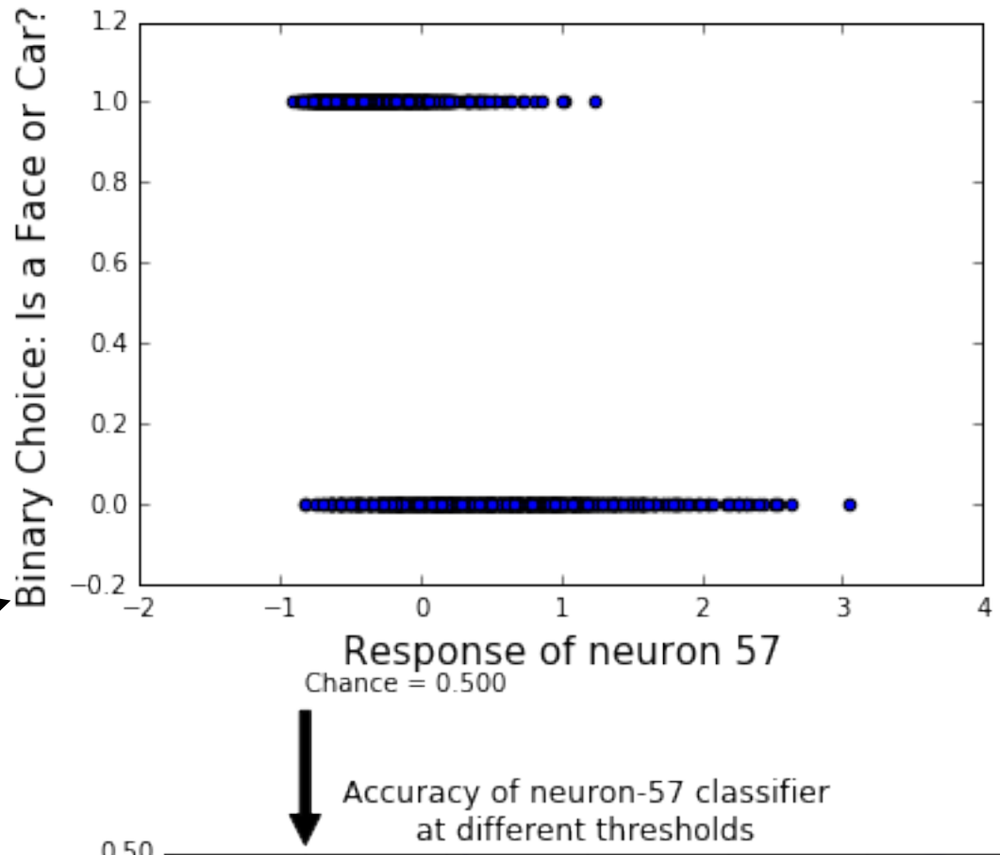
Simple correlations of neurons to stimulus properties

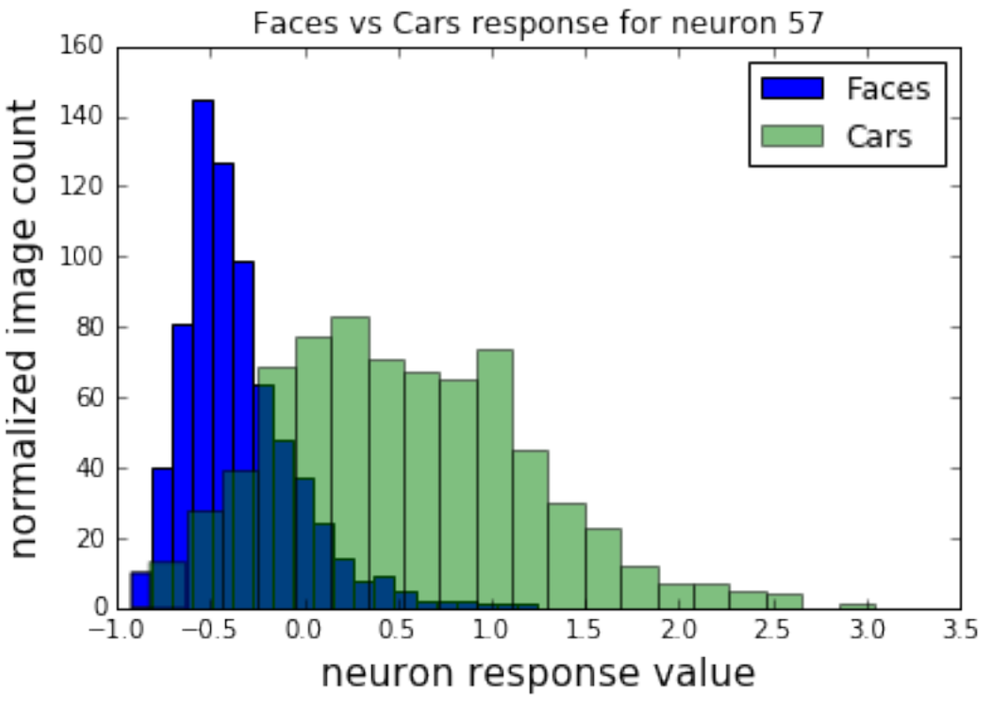


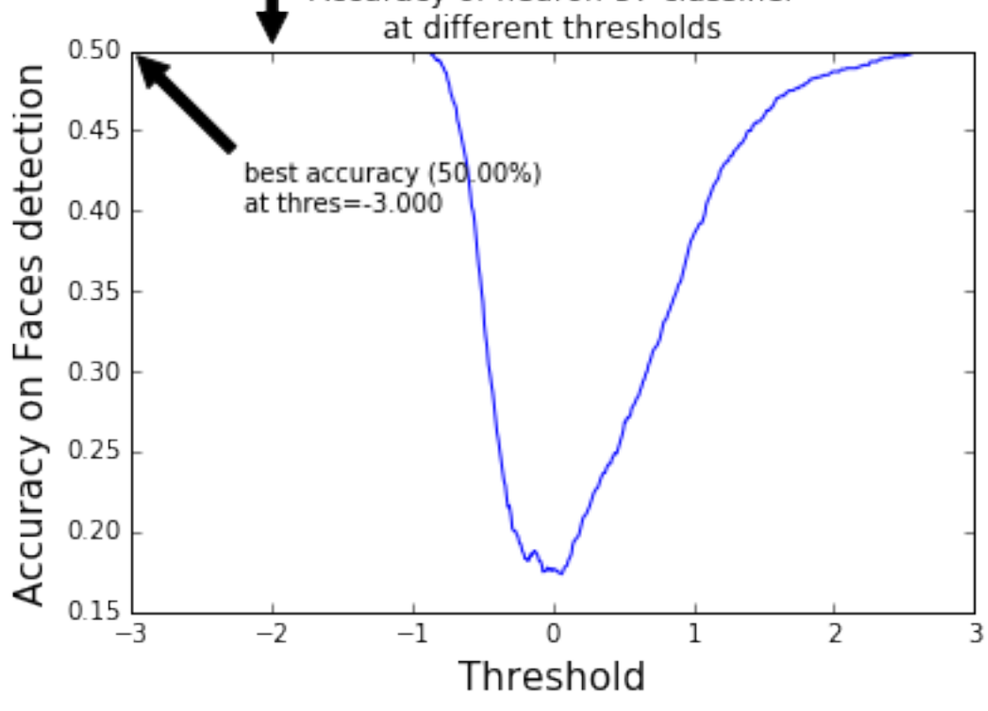
Accuracy = 73.2%

Simple correlations of neurons to stimulus properties

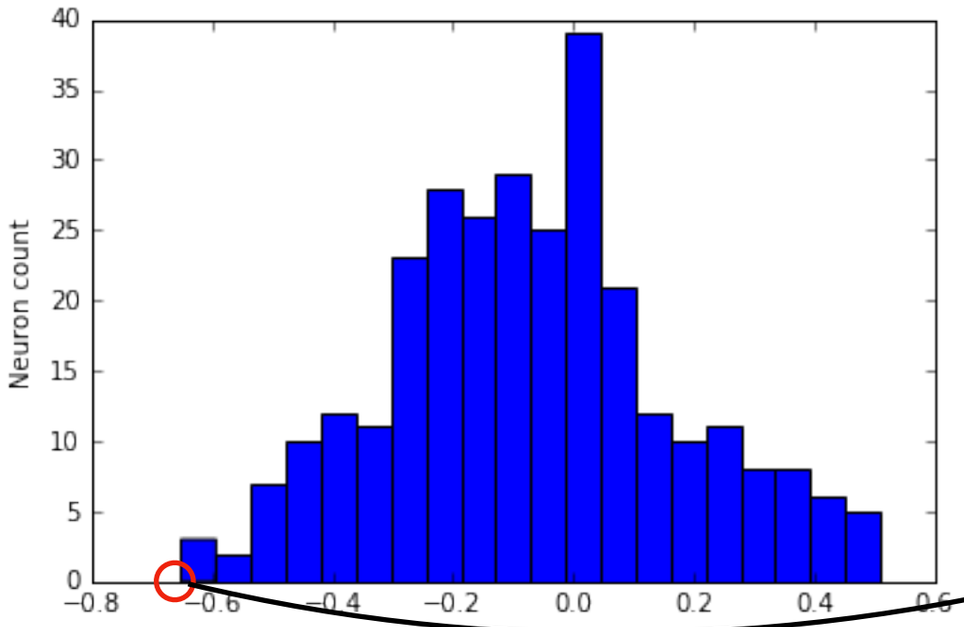


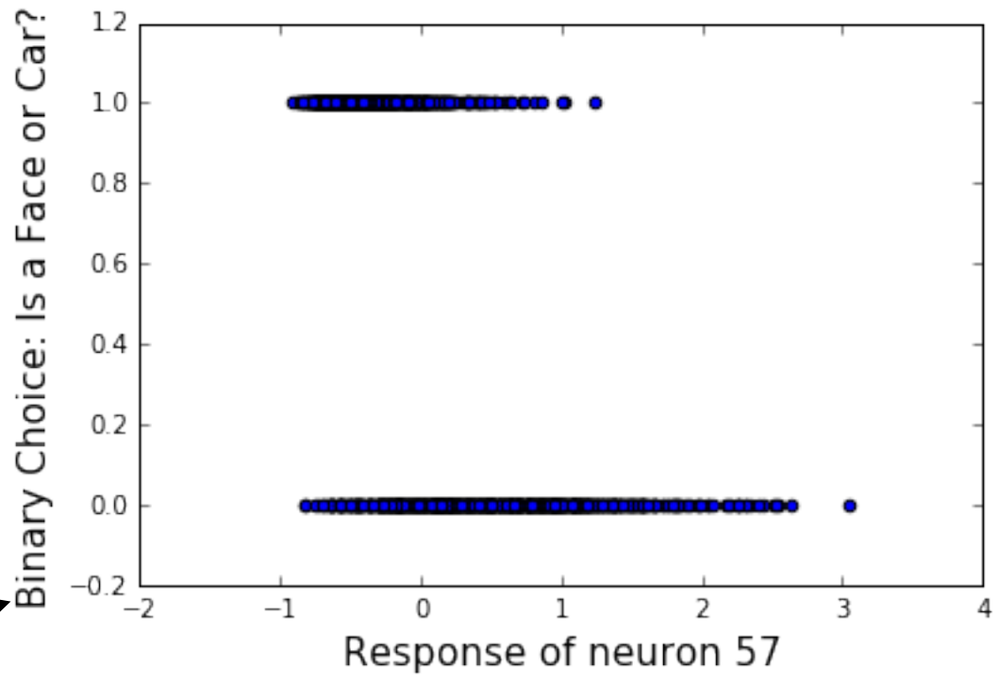


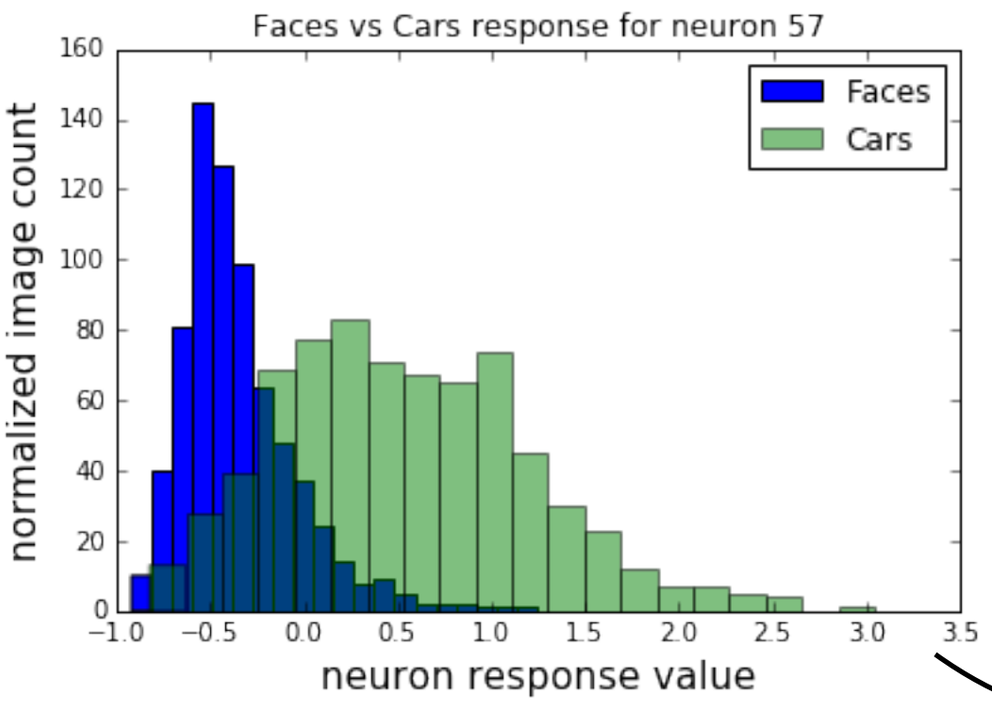




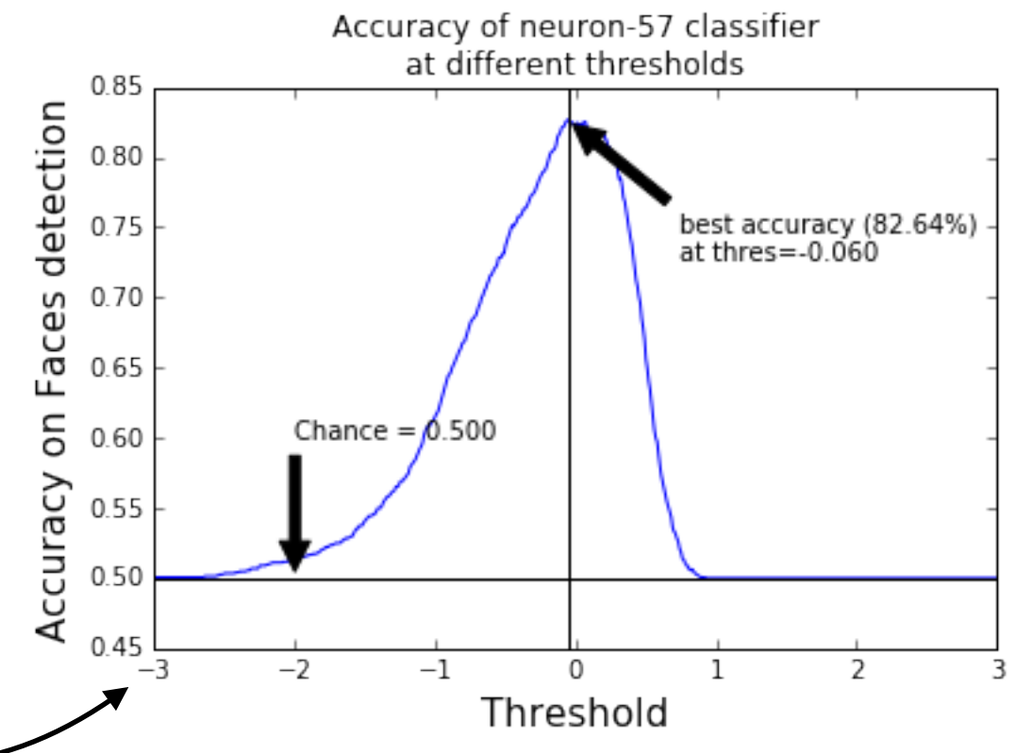
Simple correlations of neurons to stimulus properties



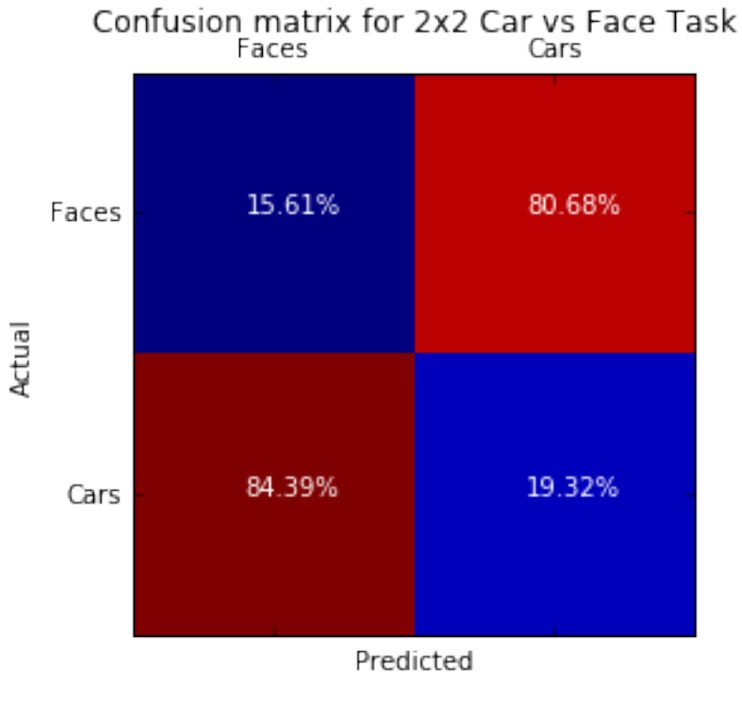


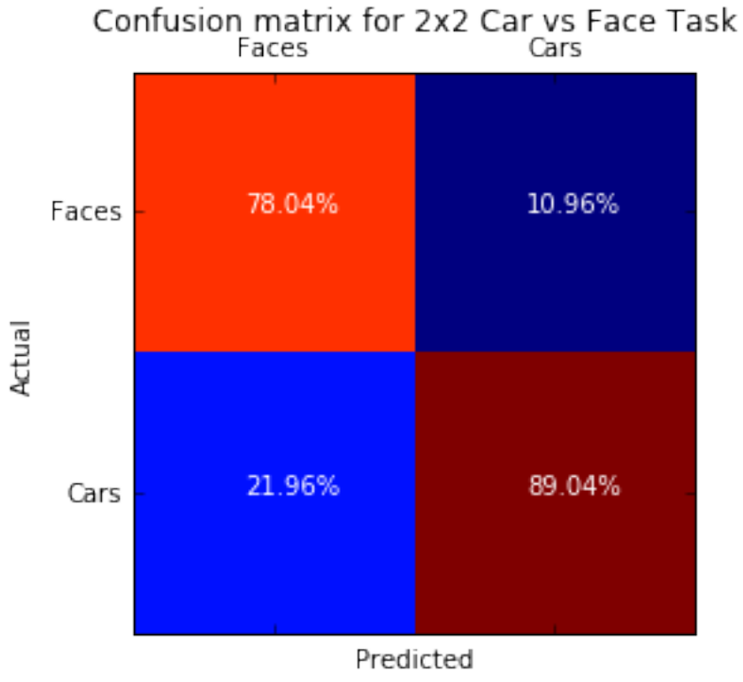


x -1



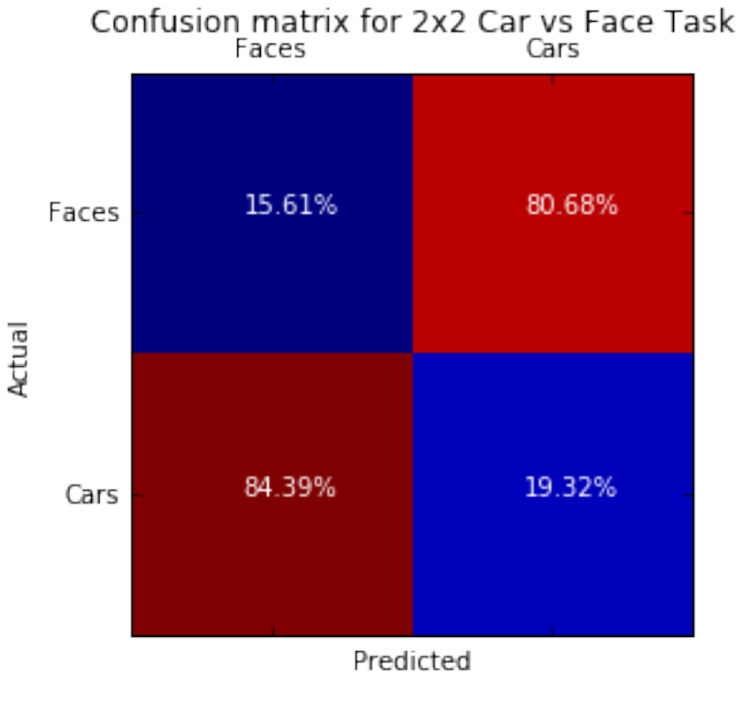
Simple correlations of neurons to stimulus properties

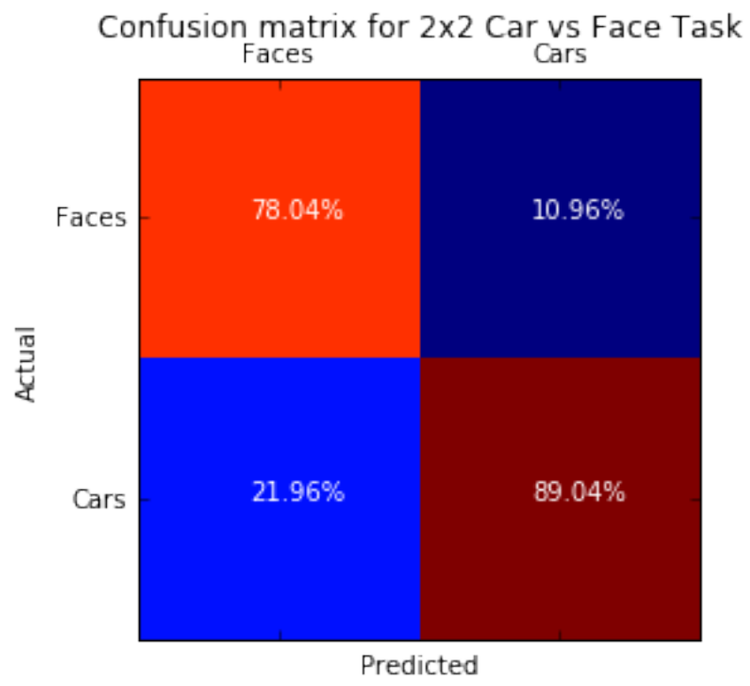




Accuracy = 82.6%

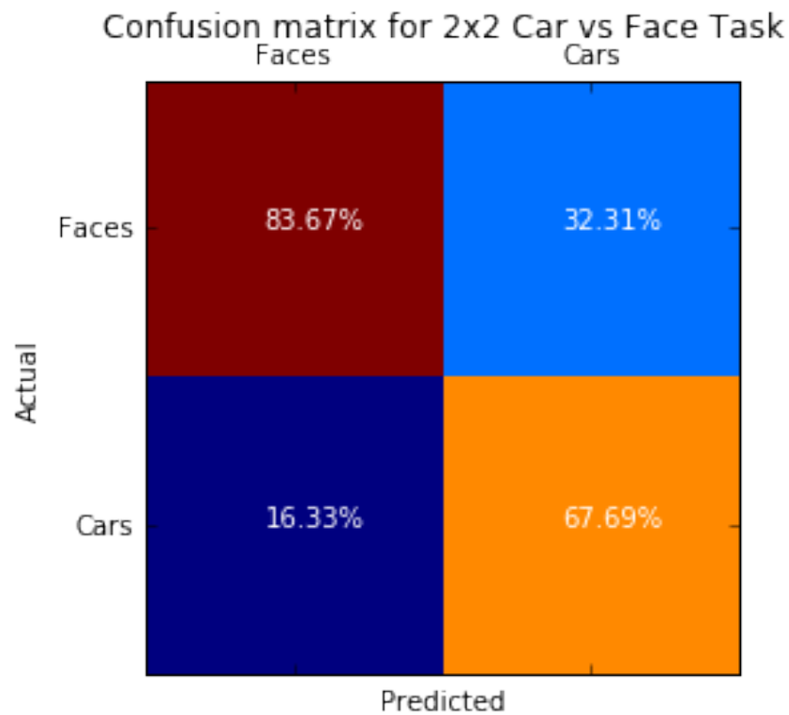
Simple correlations of neurons to stimulus properties





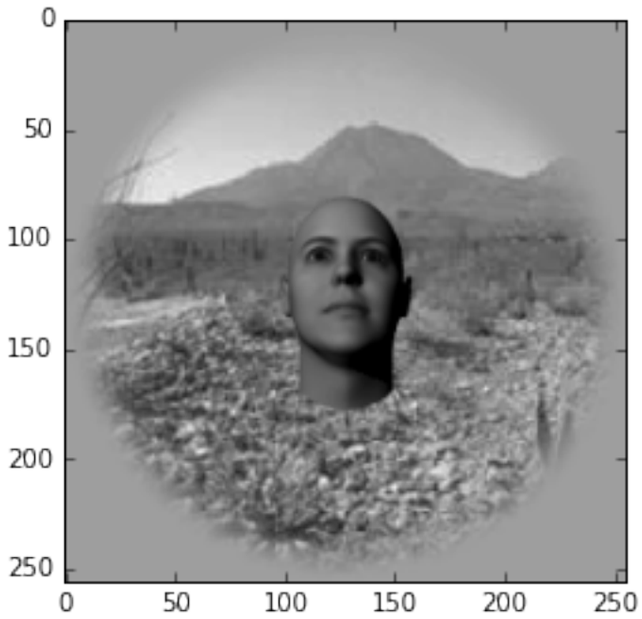
Accuracy = 82.6%

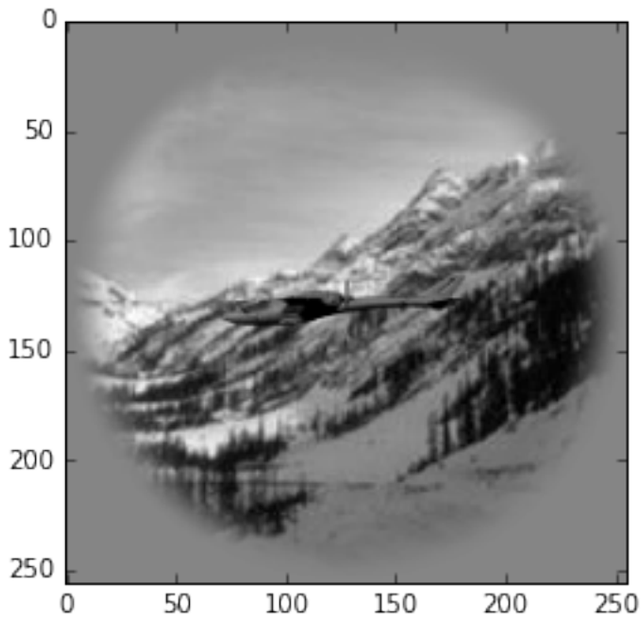
Compare to previous result:

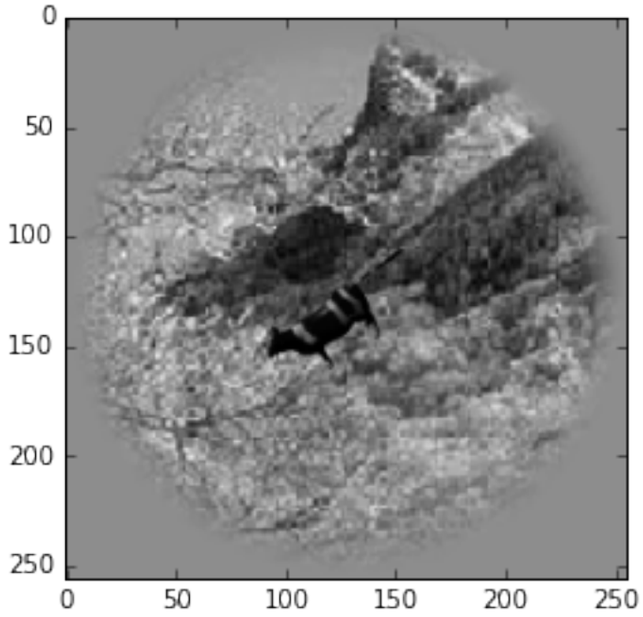


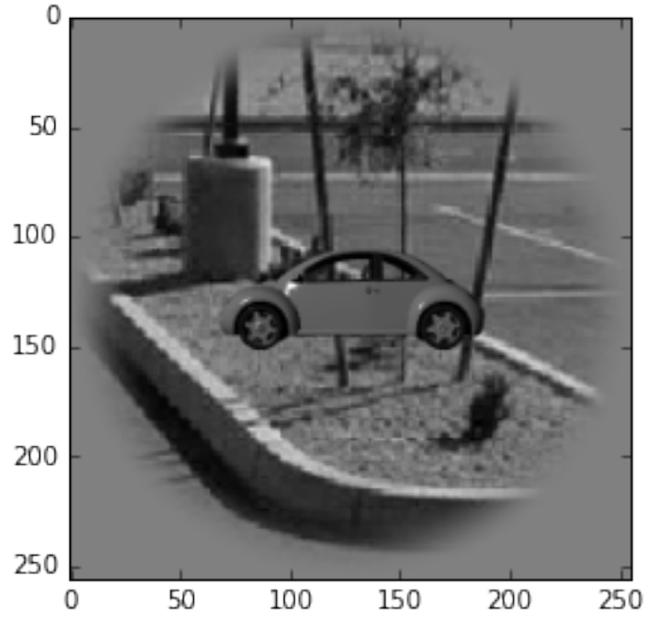
Accuracy = 73.2%

Simple correlations of neurons to stimulus properties

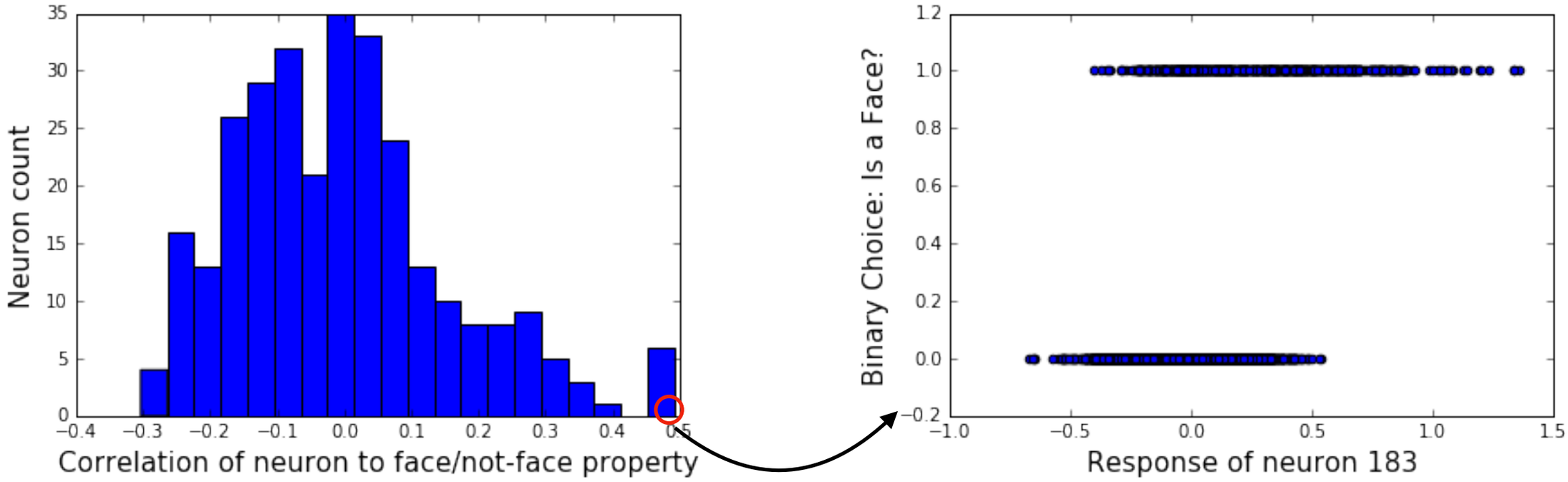


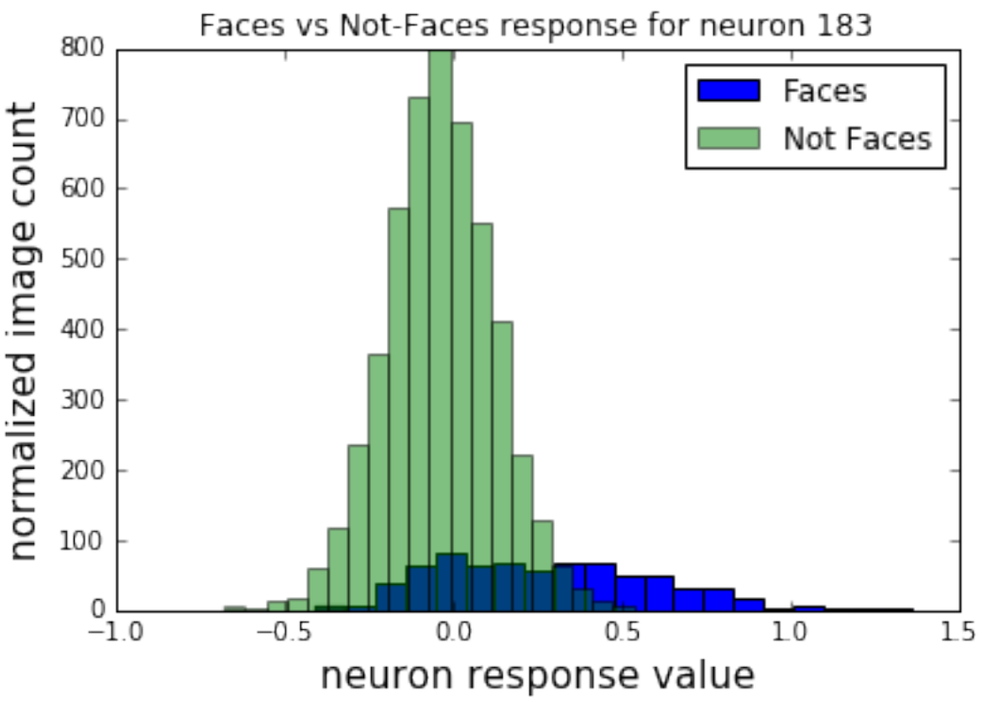


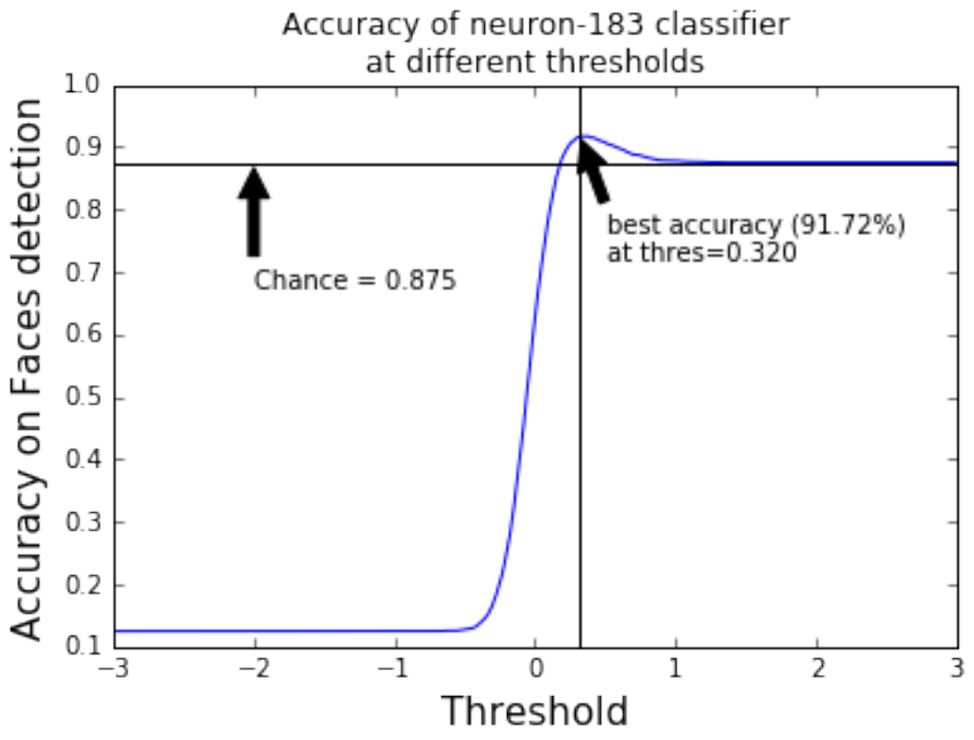




Simple correlations of neurons to stimulus properties

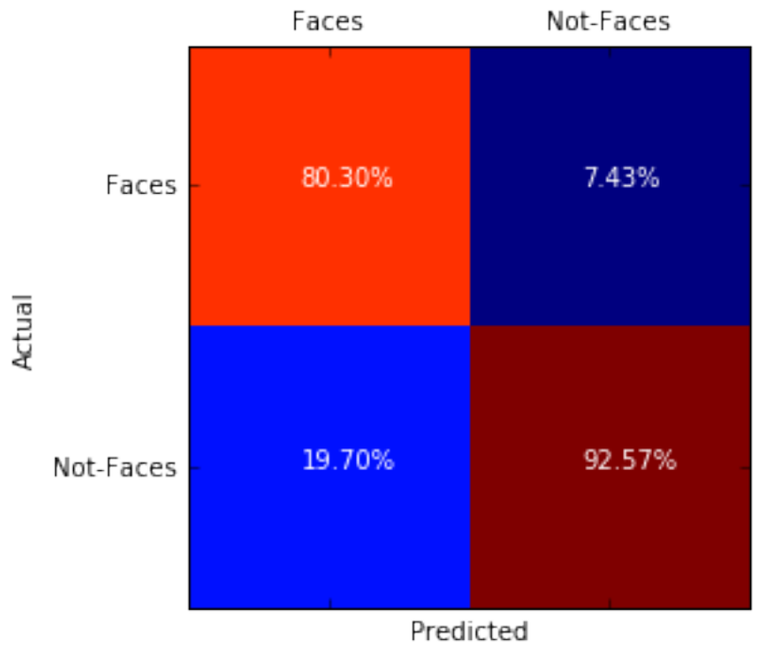






Simple correlations of neurons to stimulus properties

Face-vs-not-face

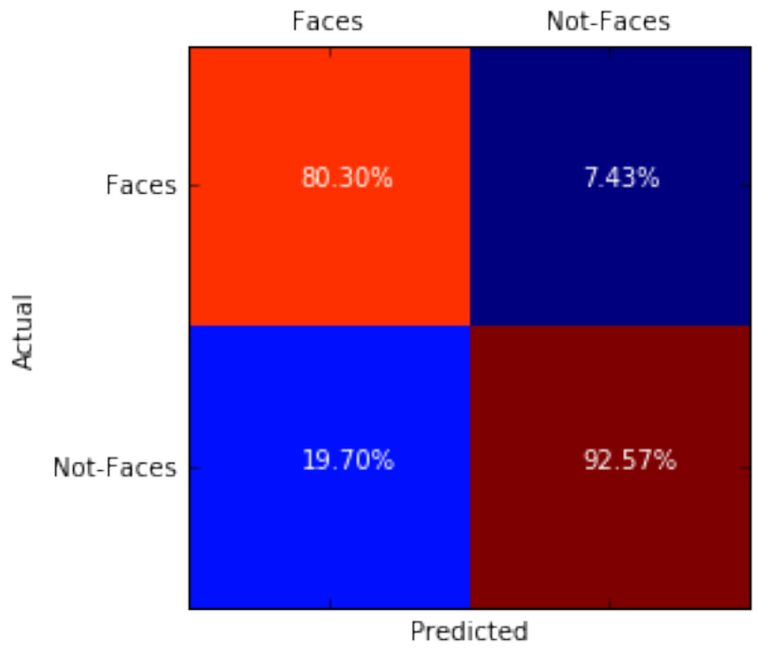


Accuracy = 91.7%

Chance = 87.5%

Simple correlations of neurons to stimulus properties

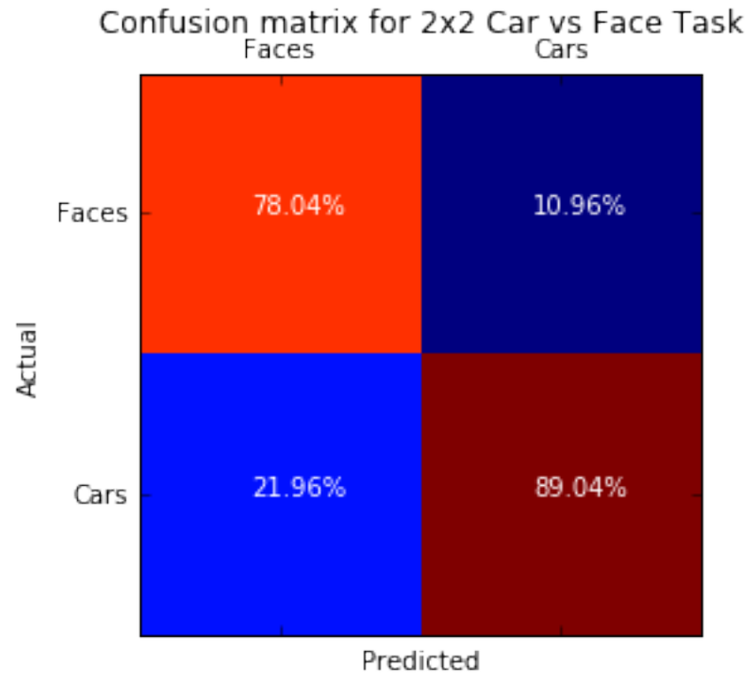
Face-vs-not-face



Accuracy = 91.7%

Chance = 87.5%

Face-vs-car



Accuracy = 82.6%

Chance = 50%

D-prime

$$d' = \frac{\mu_S - \mu_N}{\sqrt{\frac{1}{2}(\sigma_S^2 + \sigma_N^2)}}$$

μ_S = mean of signal μ_N = mean of noise

σ_S = std of signal σ_N = std of noise

D-prime

	Response: Different (*yes*)	Response: Same (*no*)
Stimuli: YES (*different*)	**HIT**	**MISS**
Stimuli: NO (*same*)	**FALSE ALARM**	**CORRECT REJECTION**

$$d' = Z(\text{true positive rate}) - Z(\text{false alarm rate})$$

where

$$Z = (\mathbf{CDF}\ \text{of gaussian})^{-1} = \mathbf{PPF}\ \text{of gaussian}$$

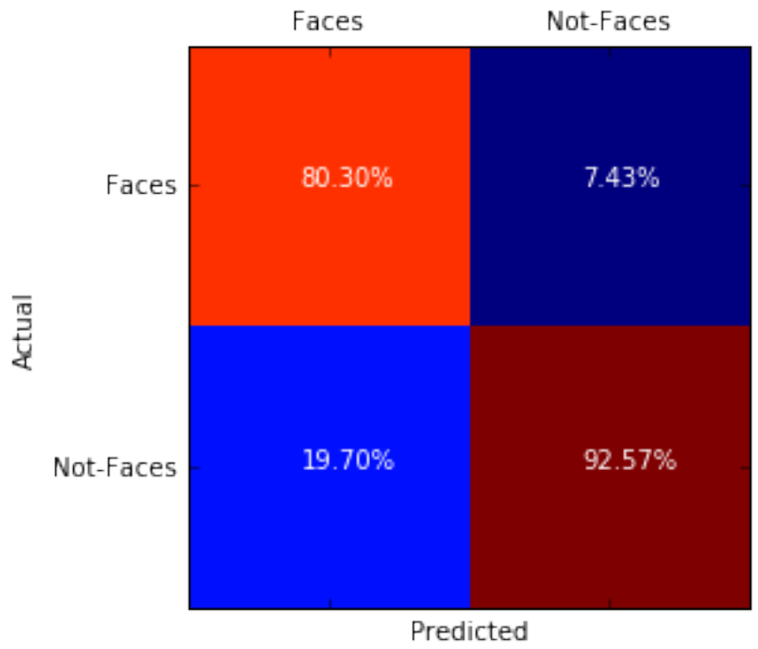
"Z-transform"

Percent-point function

http://phonetics.linguistics.ucla.edu/facilities/statistics/dprime.htm

Simple correlations of neurons to stimulus properties

Face-vs-not-face

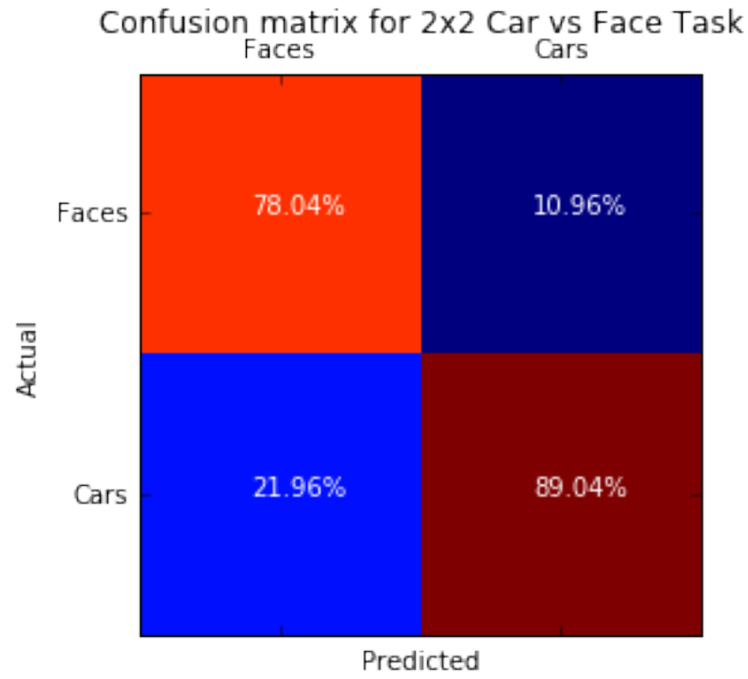


Accuracy = 91.7%

Chance = 87.5%

D-prime = 2.02

Face-vs-car



Accuracy = 82.6%

Chance = 50%

D-prime = 1.998

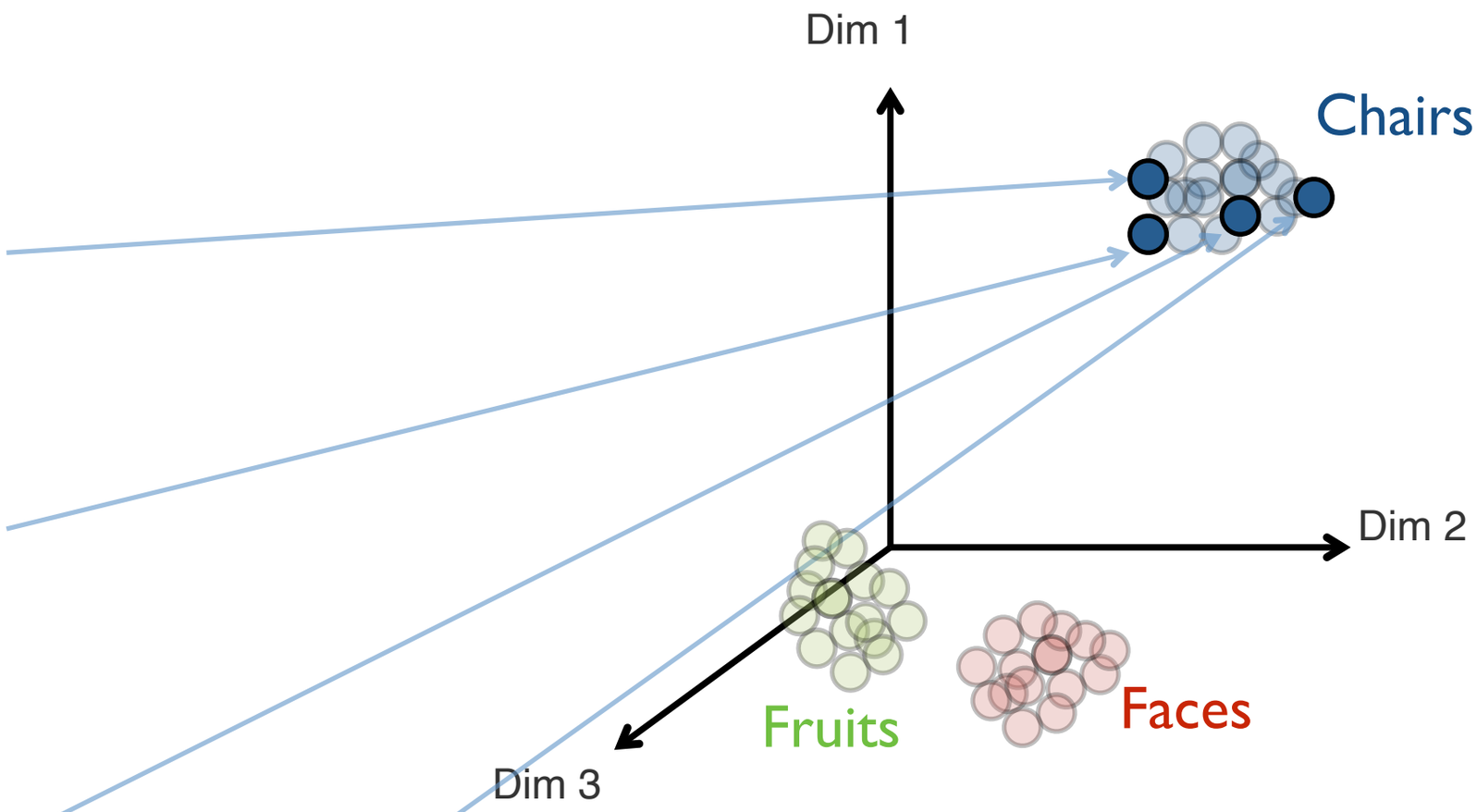
Minimum Distance Classifiers

Minimum Distance Classifiers

Pixel space: $R^{\sim 1000000}$

IT feature space: $R^{4000(?)}$

CHAIRS



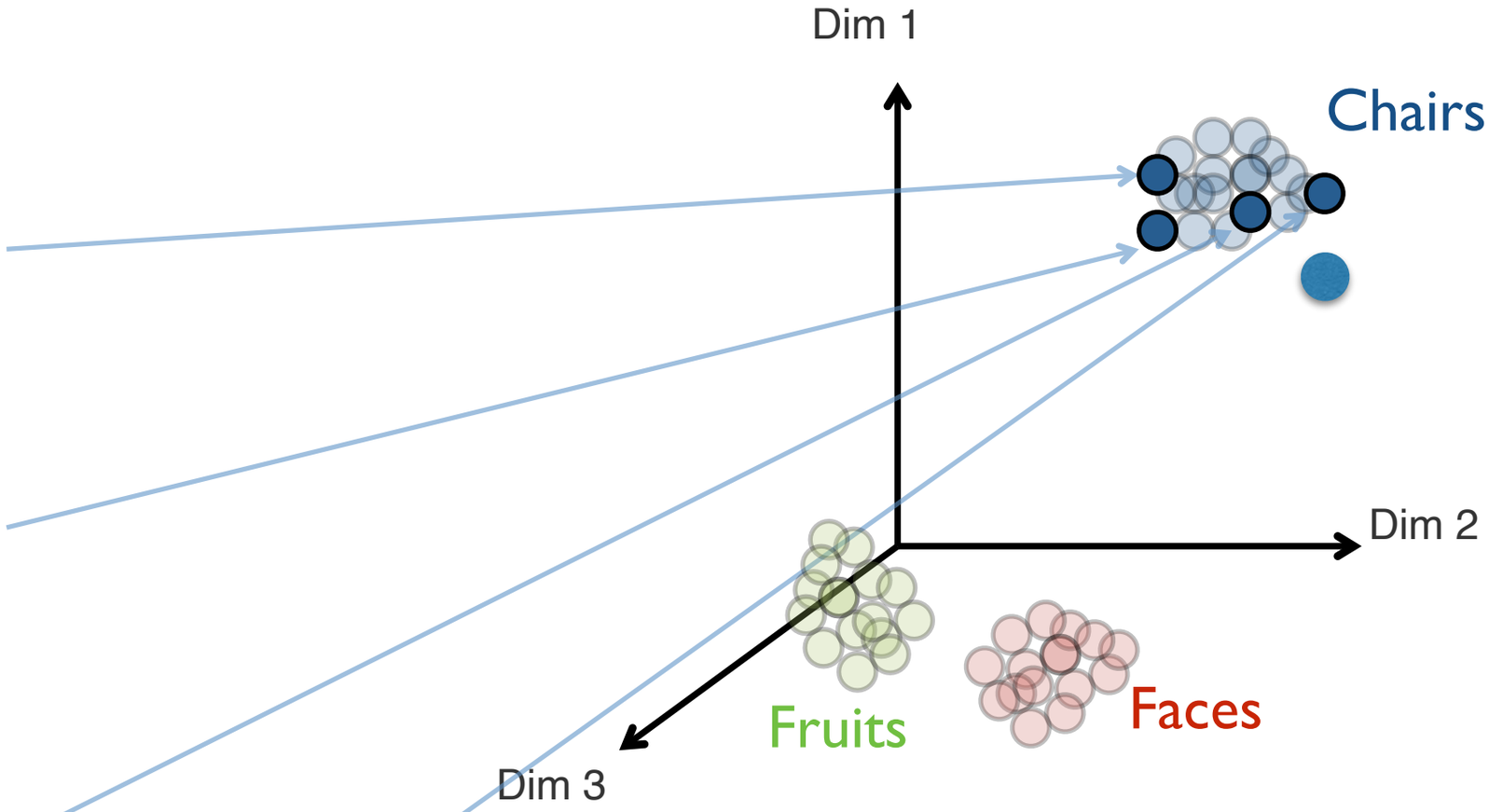
Idea of minimum distance classifiers:

For any new example, pick class whose mean is closest in feature space.

Minimum Distance Classifiers

Pixel space: $R^{\sim 1000000}$

IT feature space: $R^{4000(?)}$



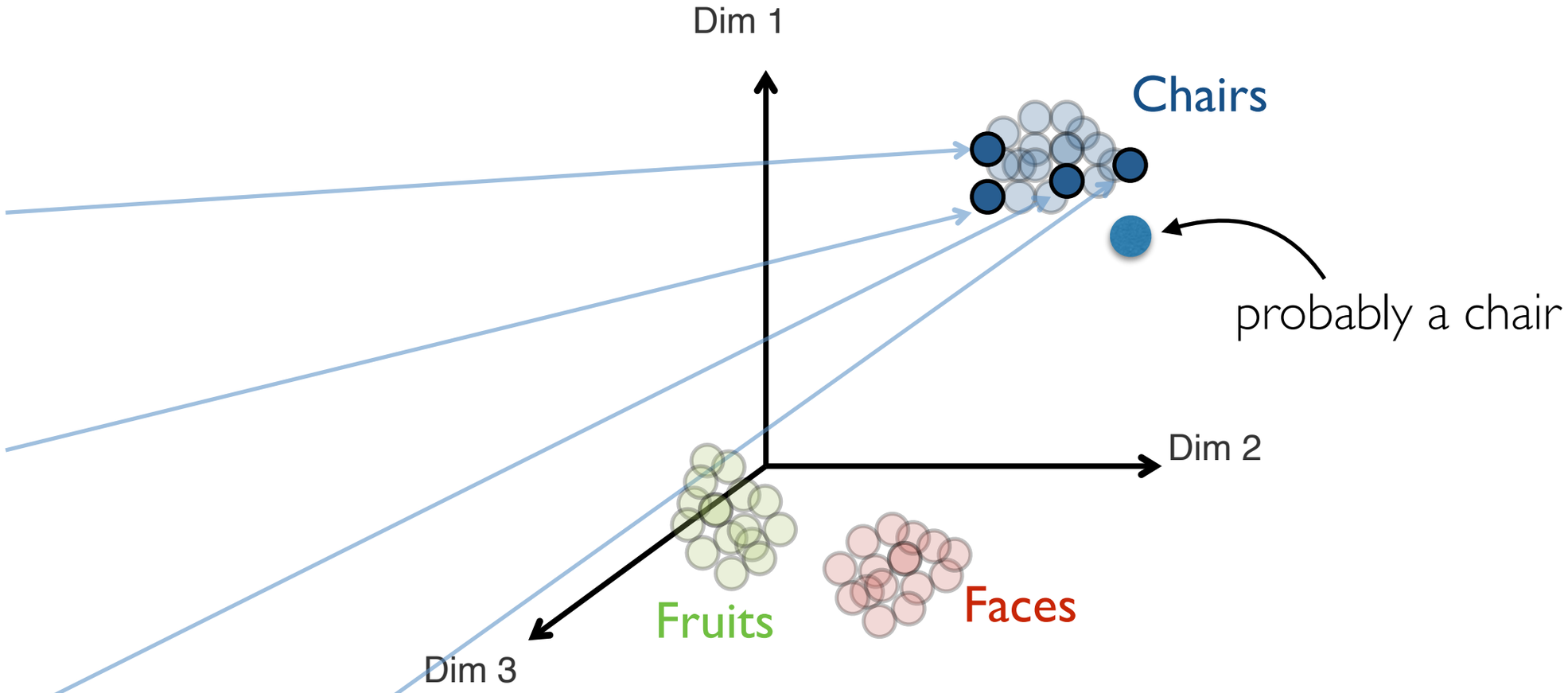
Idea of minimum distance classifiers:

For any new example, pick class whose mean is closest in neural space.

Minimum Distance Classifiers

Pixel space: $R^{\sim 1000000}$

IT feature space: $R^{4000(?)}$



Idea of minimum distance classifiers:

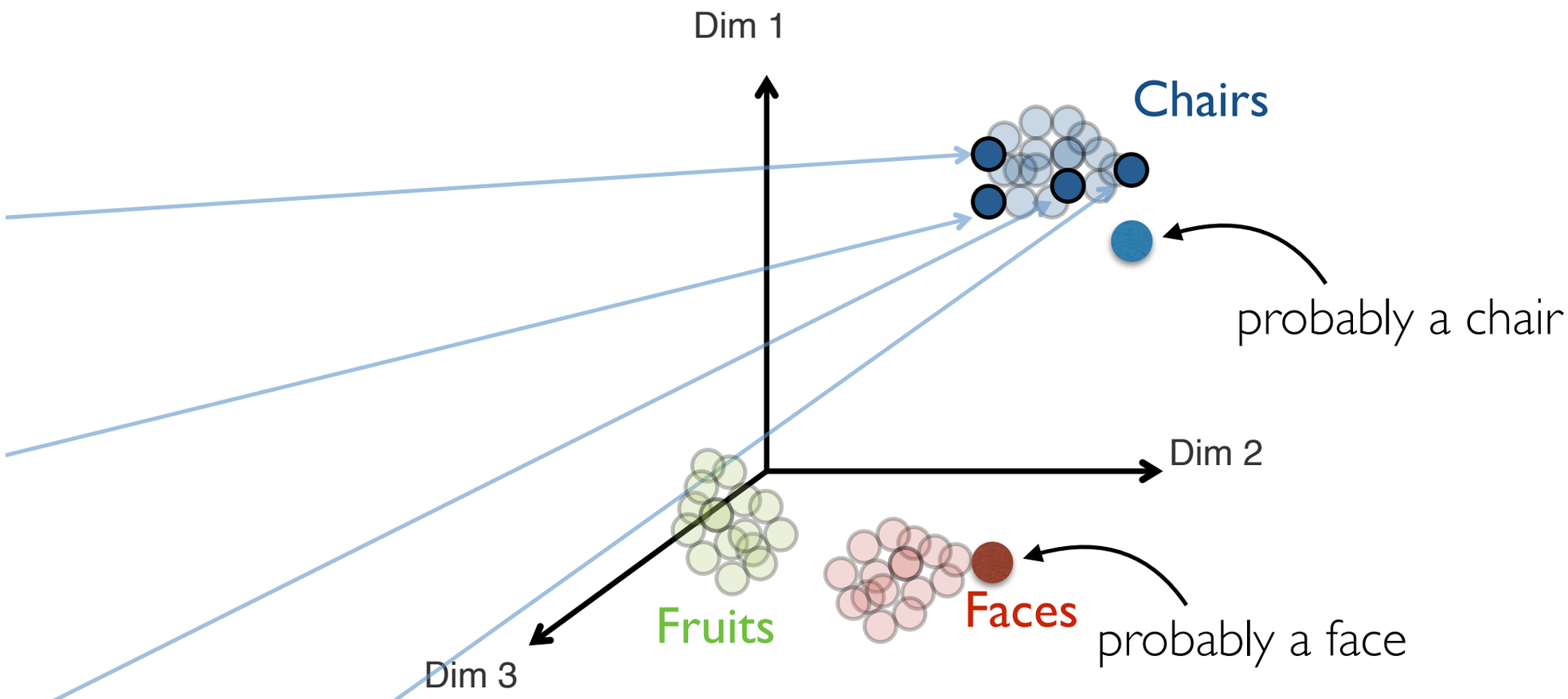
For any new example, pick class whose mean is closest in feature space.

Minimum Distance Classifiers

Pixel space: $R^{\sim 1000000}$

IT feature space: $R^{4000(?)}$

CHAIRS



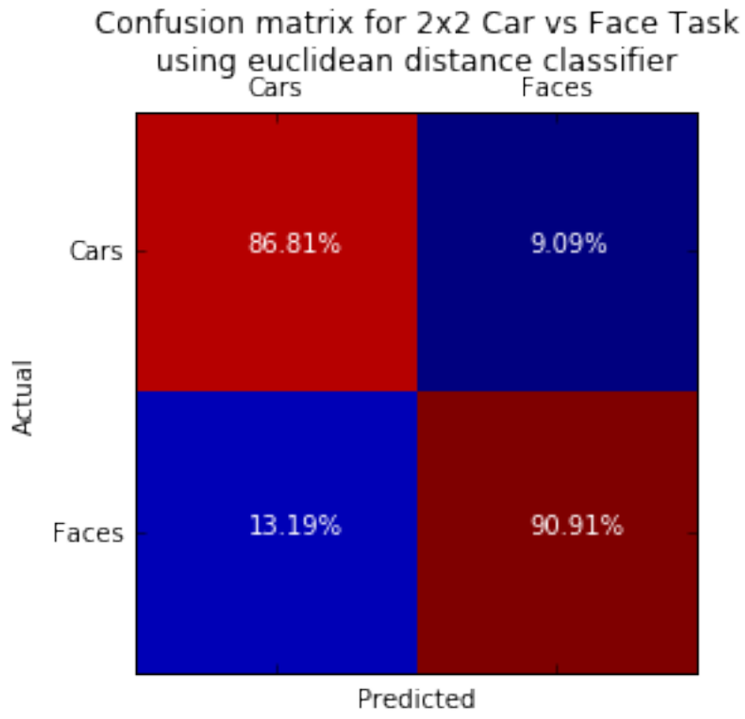
Idea of minimum distance classifiers:

For any new example, pick class whose mean is closest in neural space.

Minimum Distance Classifiers

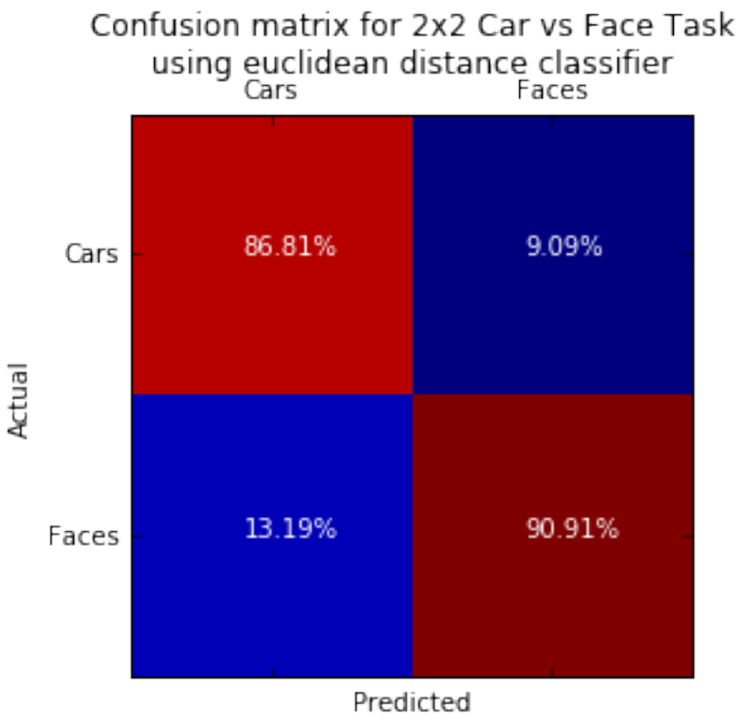
[IPYNB: Minimum Distance Classifiers]

Minimum Distance Classifiers



Simple correlations of neurons to stimulus properties

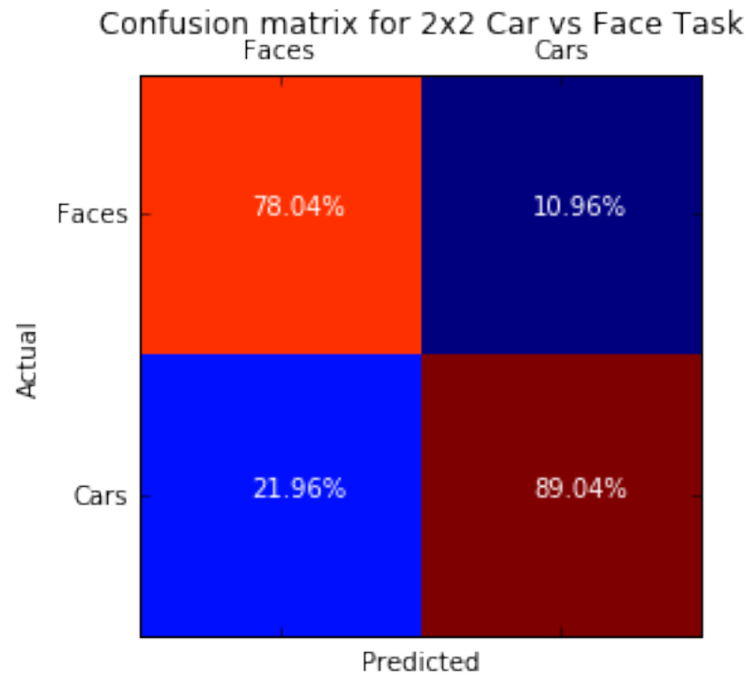
Face-vs-car
Minimum-distance classifier



Accuracy = 88.75%

Chance = 50%

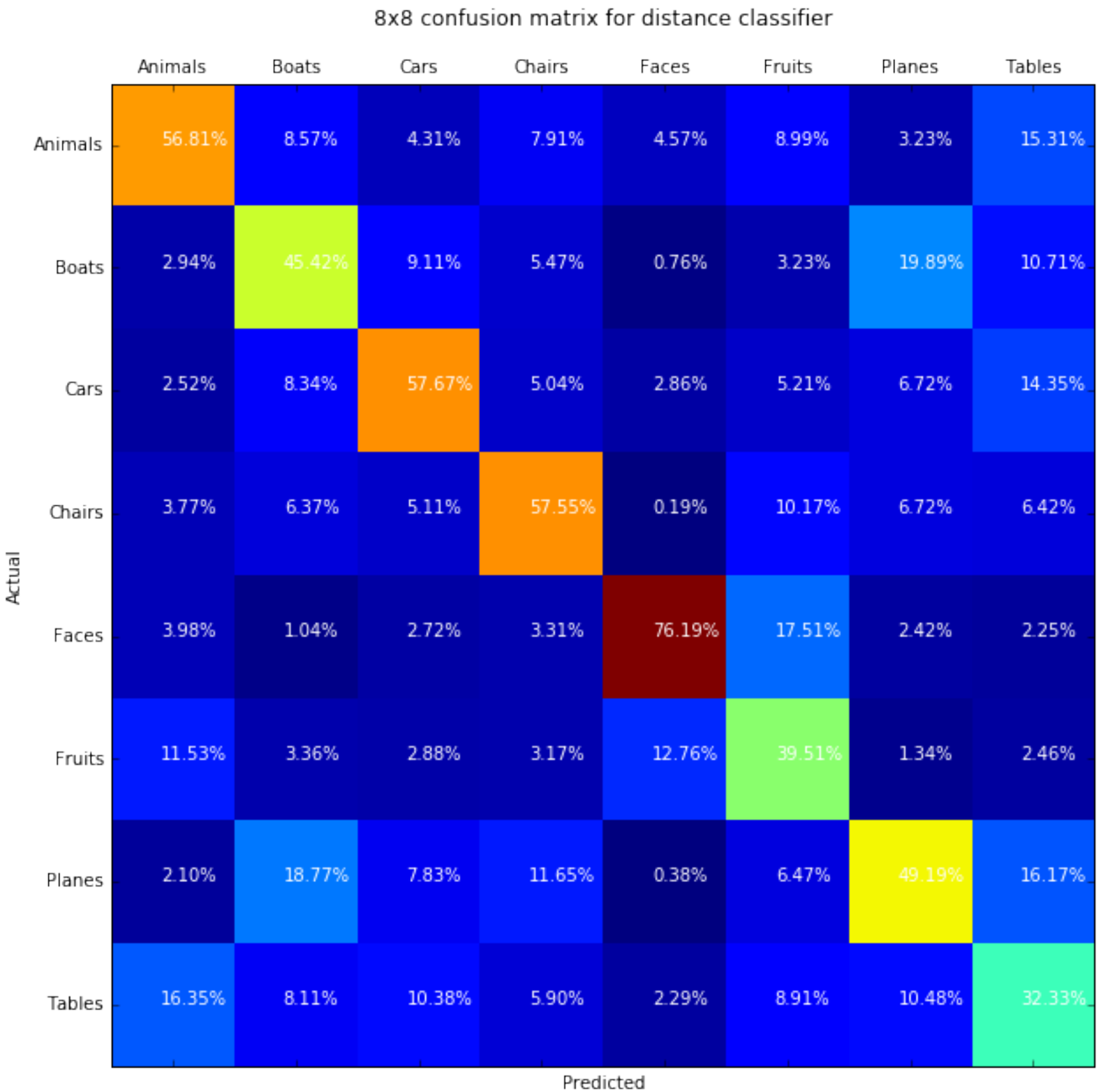
Face-vs-car
Single-best feature



Accuracy = 82.6%

Chance = 50%

Simple correlations of neurons to stimulus properties



Accuracy = 48.8%

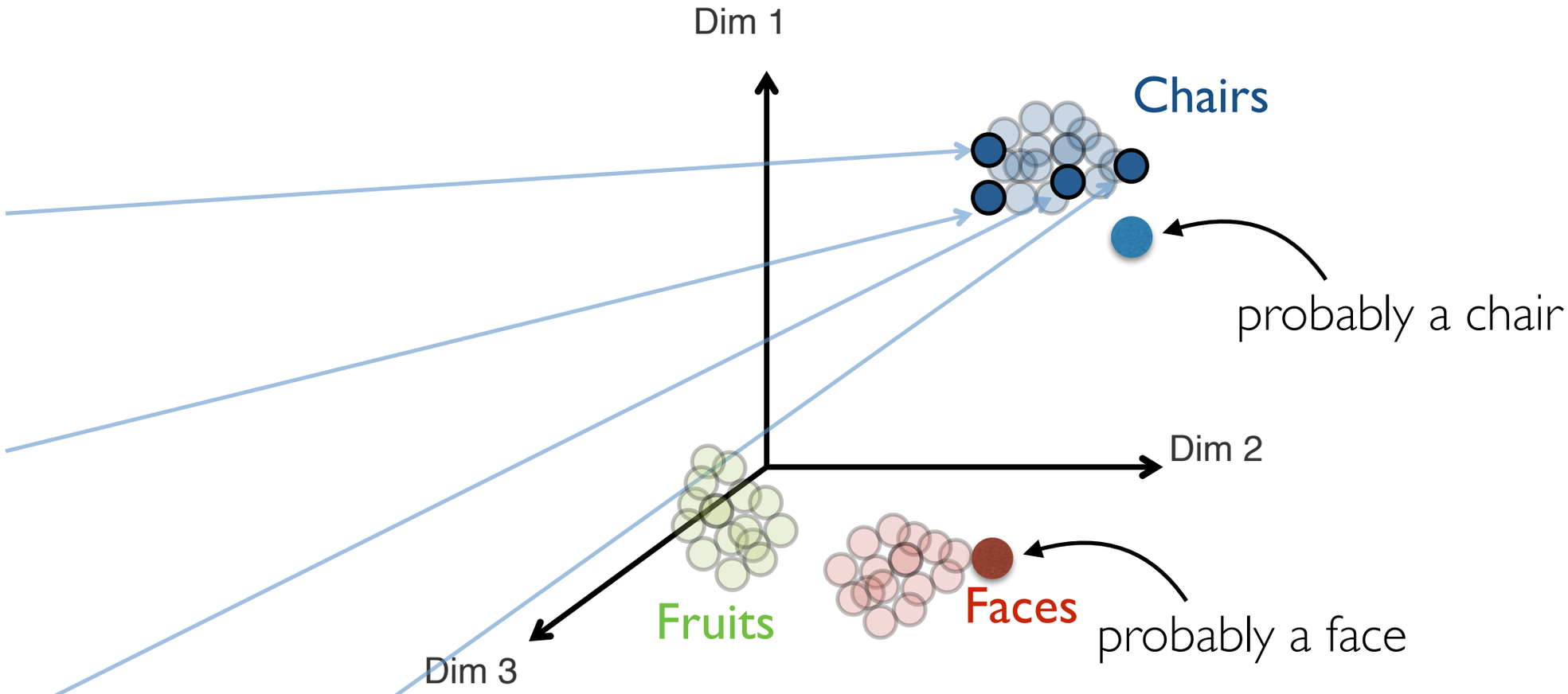
Chance = 12.5%

mean d-prime = 1.5

Neural Data as Feature Representations

Pixel space: $R^{\sim 1000000}$

IT feature space: $R^{4000(?)}$



Idea of minimum distance classifiers:

For any new example, pick class whose mean is closest in feature space.

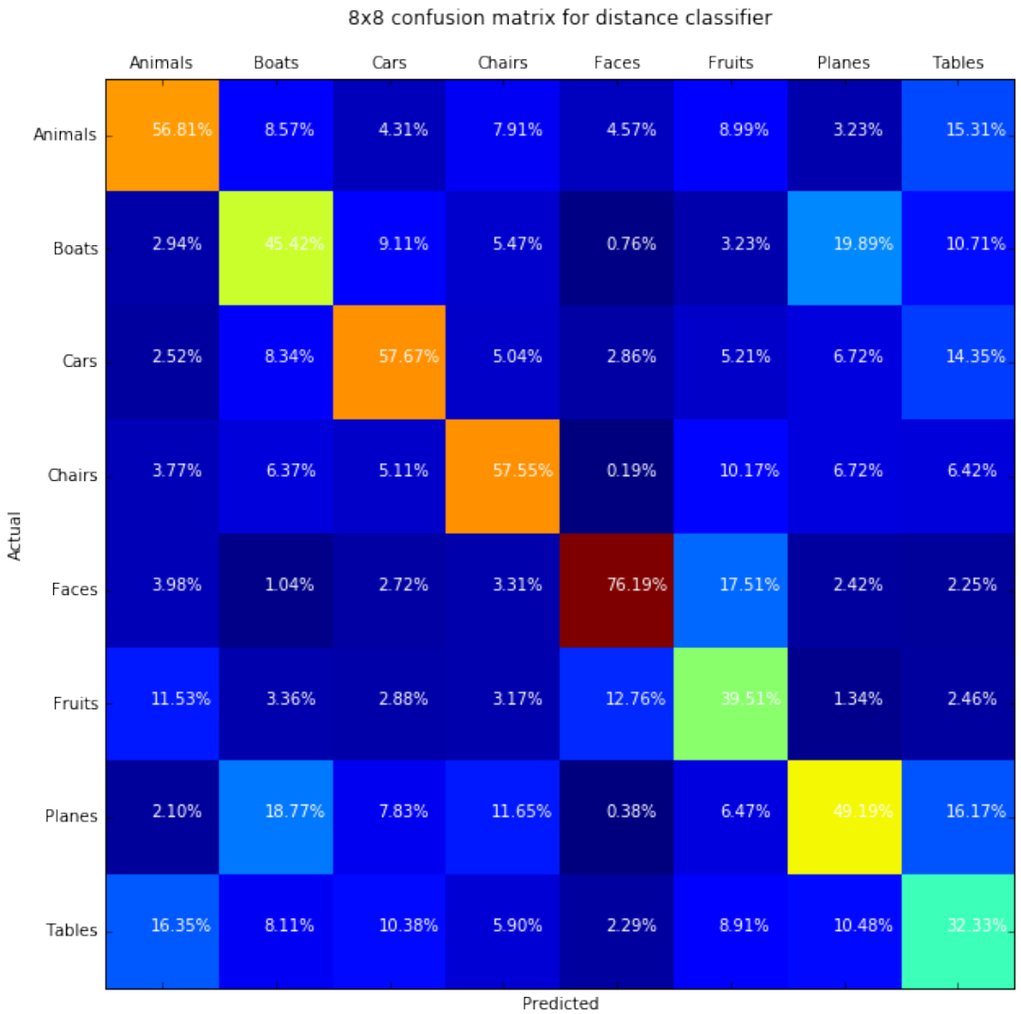
What if we used correlation distance rather than euclidean?

Minimum Distance Classifiers

[IPYNB: Maximum Correlation Classifiers]

Minimum Distance Classifiers

8-way min. Euclidean distance

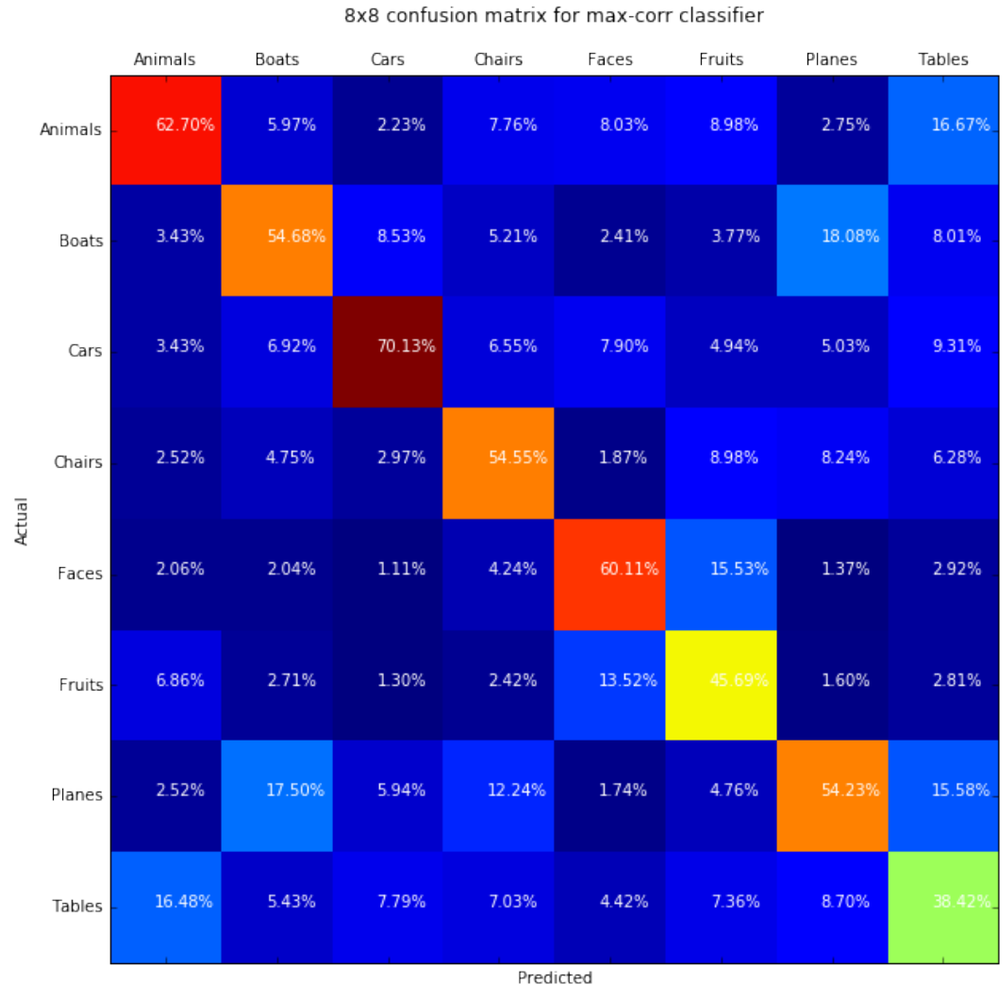


Accuracy = 48.8%

Chance = 12.5%

mean d-prime = 1.5

8-way max-correlation



Accuracy = 53.0%

Chance = 12.5%

mean d-prime = 1.6

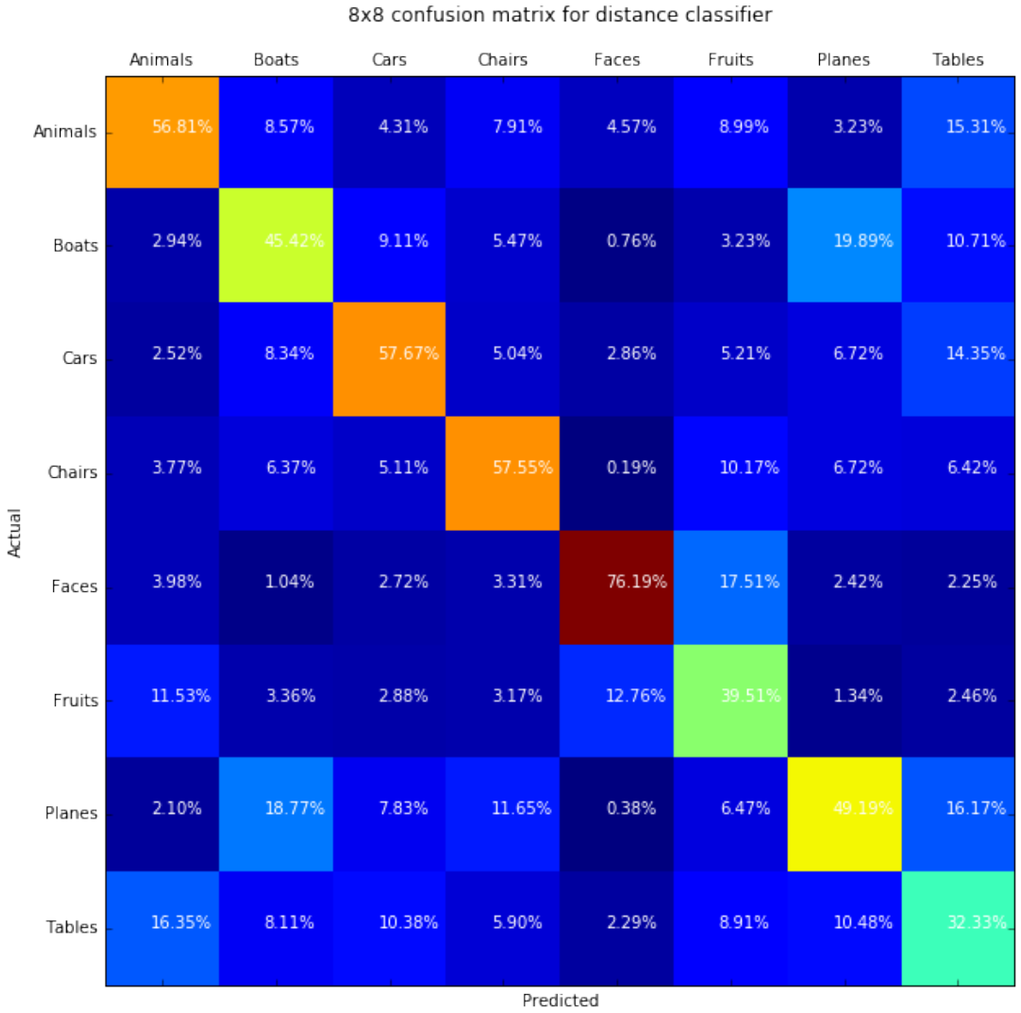
Minimum Distance Classifiers

Correlation Classifier seems a little better than the euclidean distance classifier

Norming the data helps a little as well.

Minimum Distance Classifiers

8-way min. Euclidean distance

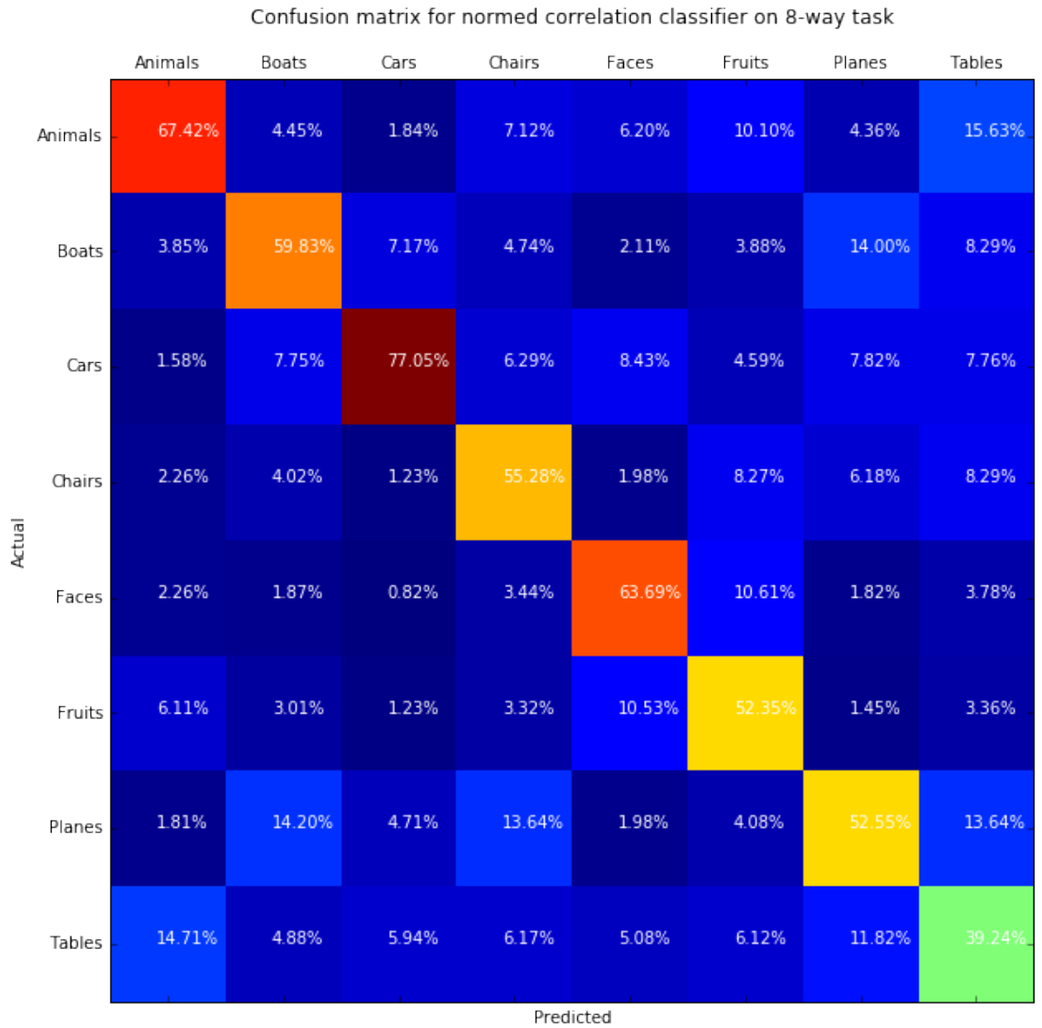


Accuracy = 48.8%

Chance = 12.5%

mean d-prime = 1.5

8-way max-correlation normed



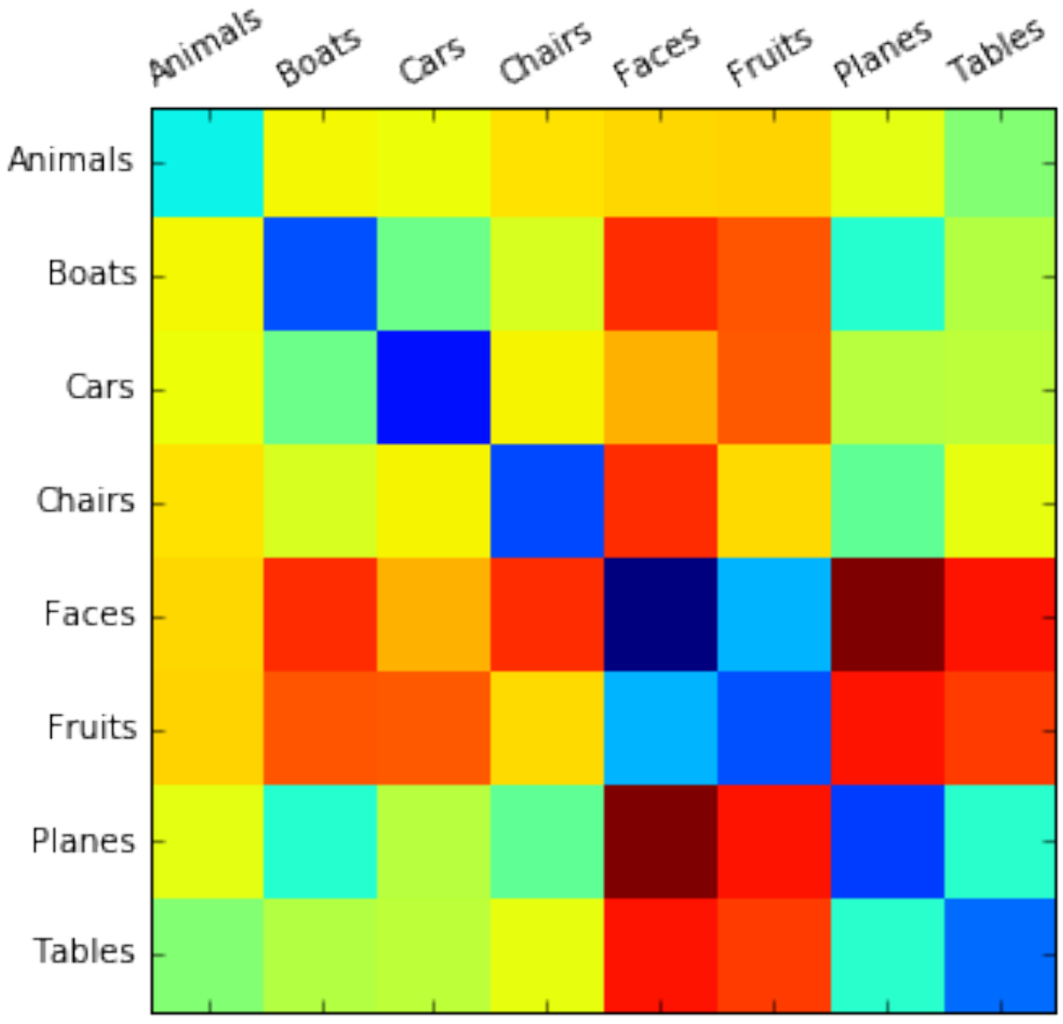
Accuracy = 56.3%

Chance = 12.5%

mean d-prime = 1.75

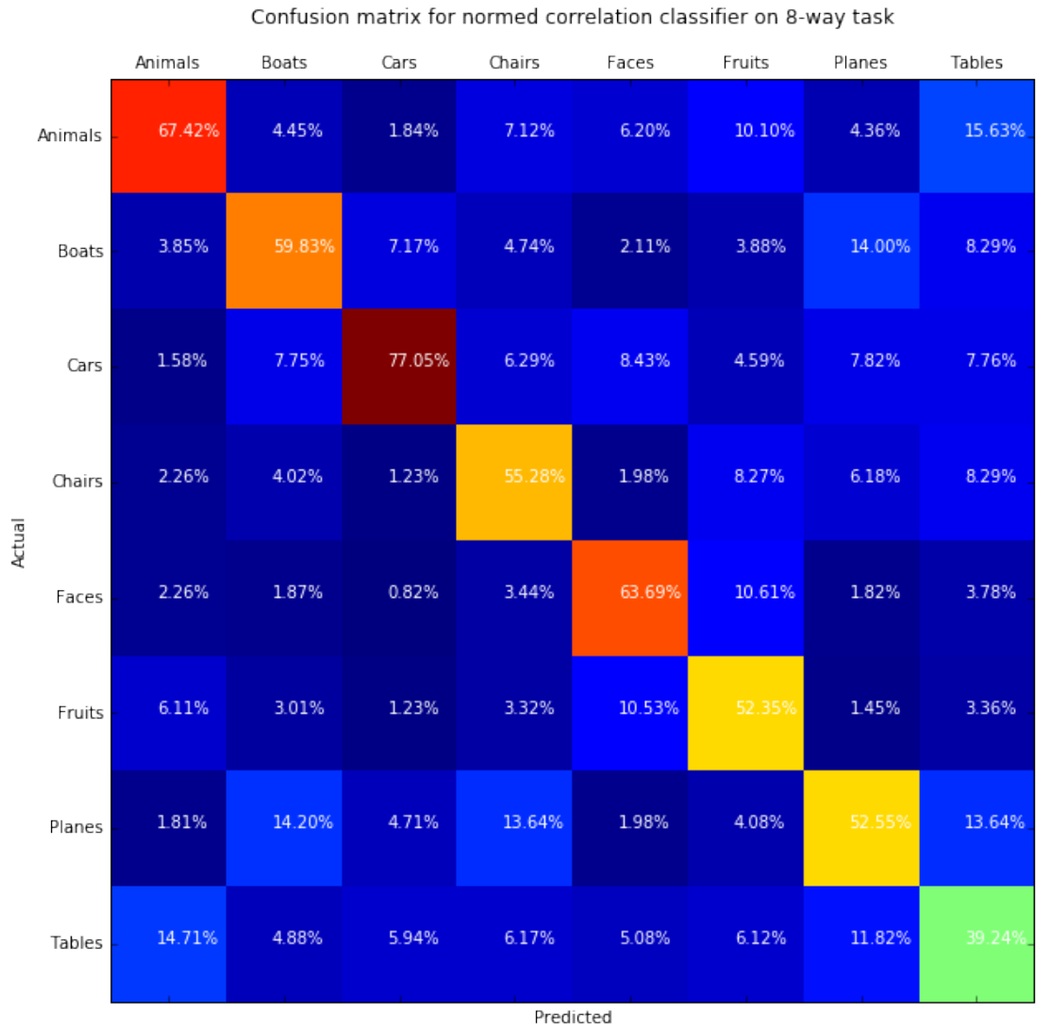
Minimum Distance Classifiers

Correlation matrix



No feature-specific weighting

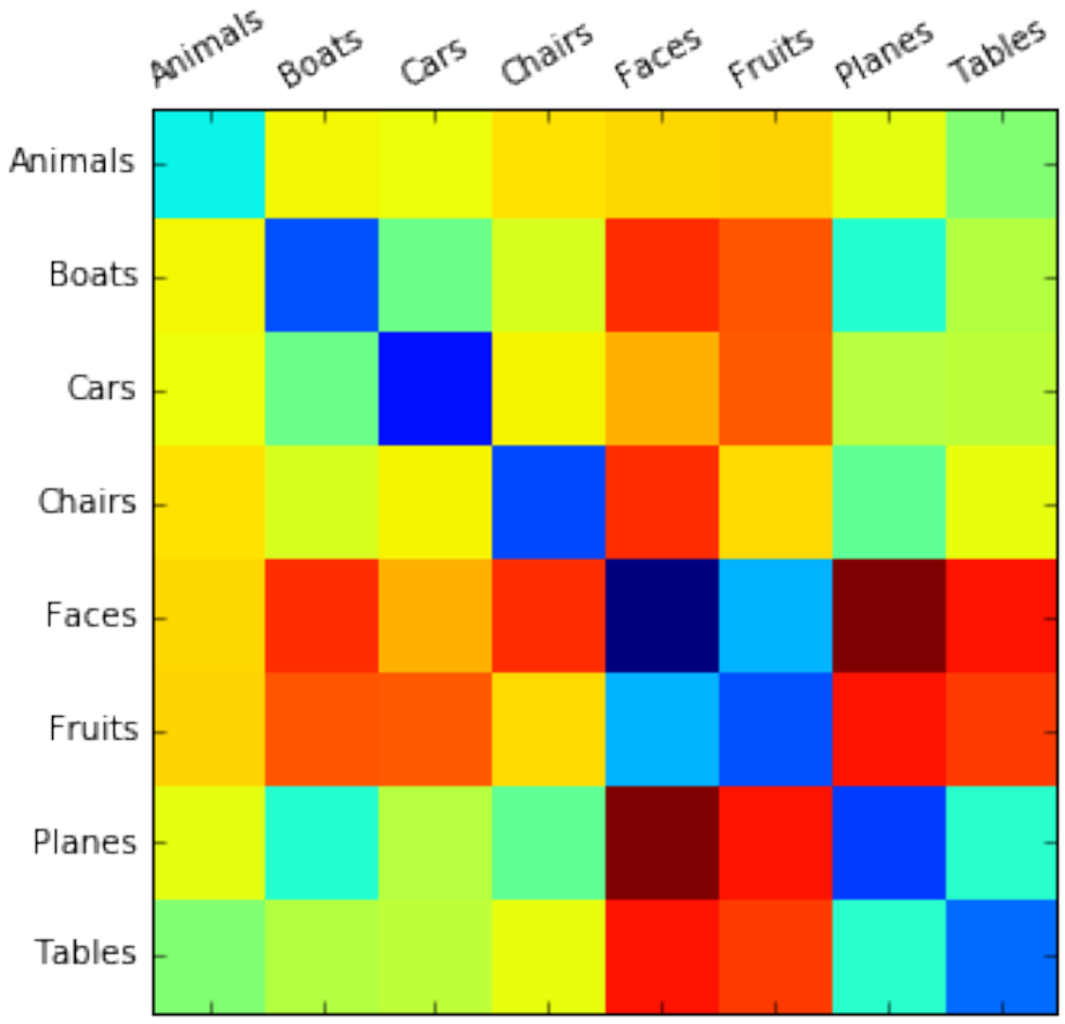
Confusion matrix



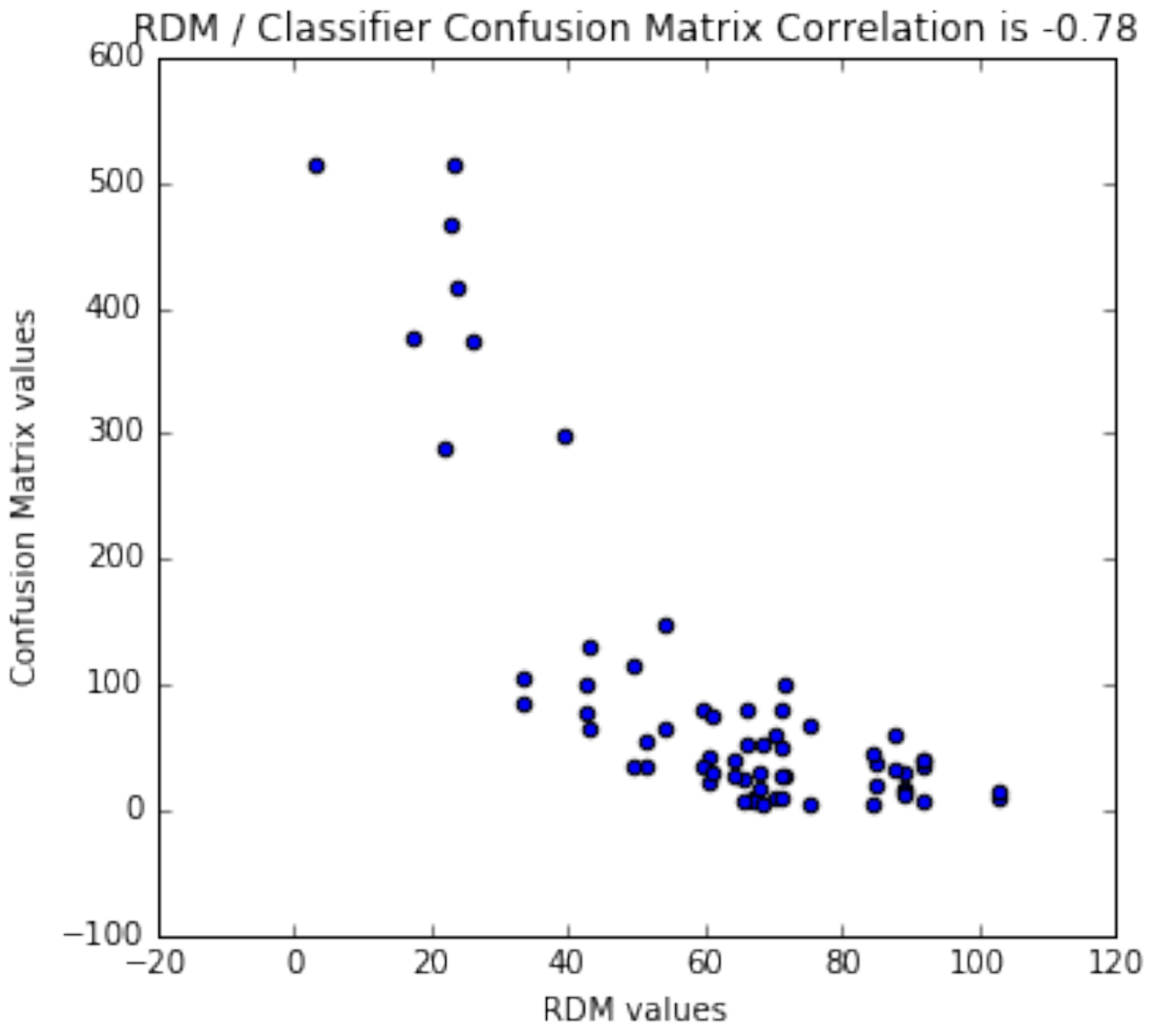
Uses correlation-based weighting

Minimum Distance Classifiers

Correlation matrix



Confusion matrix vs correlation matrix

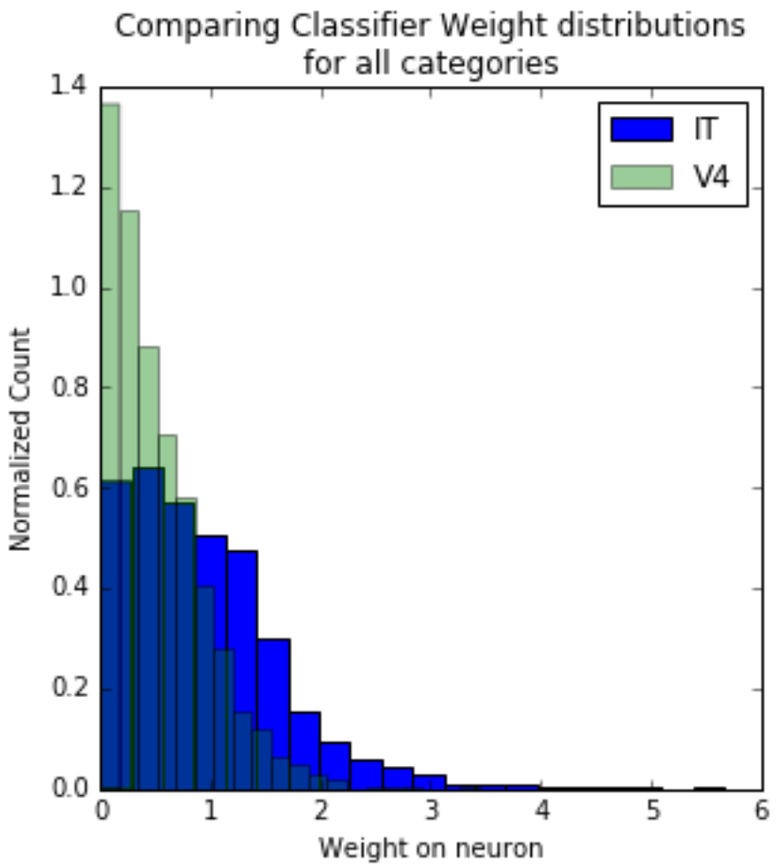


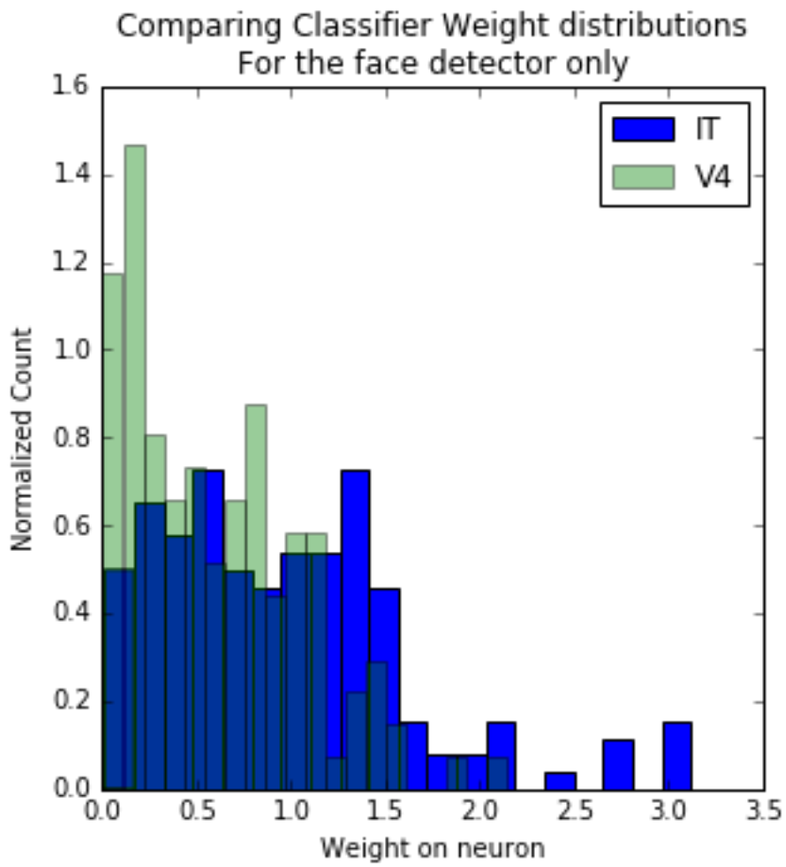
Minimum Distance Classifiers

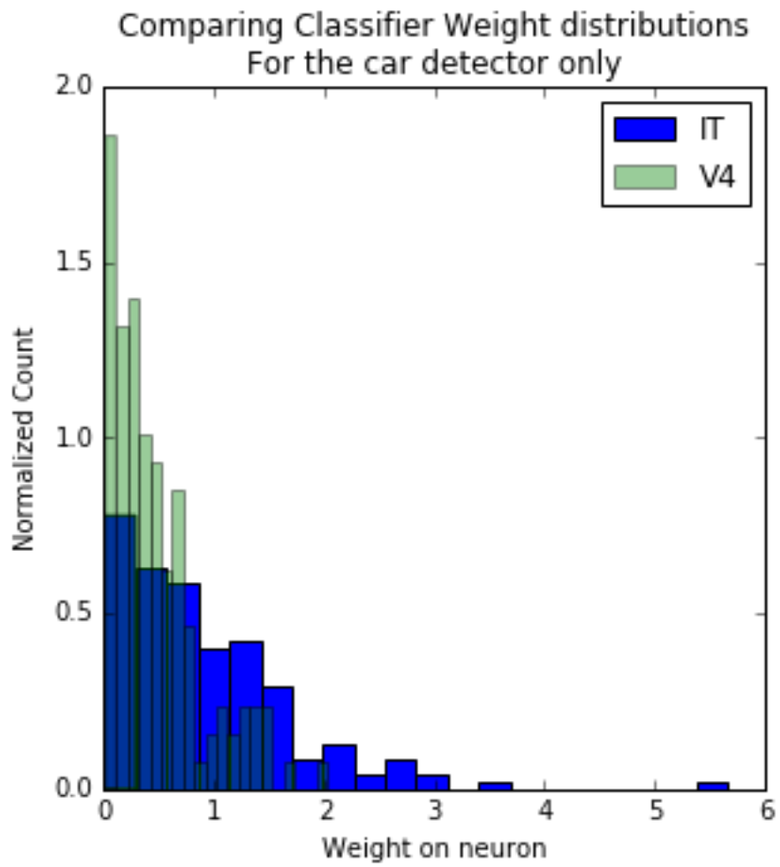
[IPYNB: Comparison of V4 and IT]

Minimum Distance Classifiers

IT neurons are doing most of the work in our classifiers.







Cross Validation

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

"Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set."

Cross Validation

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

"Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set."

```
Procedure:
    1) estimate model params on training data
    2) evaluate performance on testing data
```

Cross Validation

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

"Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set."

```
Procedure:
    1) estimate model params on training data
    2) evaluate performance on testing data
```

Q: Why are we doing this?

Cross Validation

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

"Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set."

```
Procedure:
    1) estimate model params on training data
    2) evaluate performance on testing data
```

Q: Why are we doing this?

A: To prevent overfitting (fooling ourselves)

Cross Validation

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

"Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set."

```
Procedure:
    1) estimate model params on training data
    2) evaluate performance on testing data
```

Q: Why are we doing this?

A: To prevent overfitting (fooling ourselves)

Q: How do you get the train/test splits?

Cross Validation

Q: How do you get the train/test splits?

1. Leave-***p***-out: train on all-but-***p***, test on ***p*** for all subsets of ***p*** stimuli; average results

Cross Validation

Q: How do you get the train/test splits?

1. Leave-***p***-out: train on all-but-***p***, test on ***p*** for all subsets of ***p*** stimuli; average results

2. ***k***-fold: partition data into ***k*** equal-sized subsets; train on all but one subset; test on held-out slice; average results

Cross Validation

Q: How do you get the train/test splits?

1. Leave-***p***-out: train on all-but-***p***, test on ***p*** for all subsets of ***p*** stimuli; average results

2. ***k***-fold: partition data into ***k*** equal-sized subsets; train on all but on subset; test on held-out slice; average results

3. random subsampling: randomly choose training subset and non-overlapping testing subset; average results over many random choices

Cross Validation

Q: How do you get the train/test splits?

1. Leave-***p***-out: train on all-but-***p***, test on ***p*** for all subsets of ***p*** stimuli; average results

2. ***k***-fold: partition data into ***k*** equal-sized subsets; train on all but on subset; test on held-out slice; average results

3. random subsampling: randomly choose training subset and non-overlapping testing subset; average results over many random choices

key thing in all cases: training/test CANNOT overlap!

Cross Validation

[IPYNB: Getting Splits for Cross-Validation]

Cross Validation

```
Procedure:

    splits = get_splits(split arguments)

    For each split:

      estimate model params on training data

      evaluate performance on testing data

average results over splits
```

Object-Oriented Programming (OOP)

```
class Baz(object):
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def foo(self, x):
        return x**2 / self.a
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def foo(self, x):
        return x**2 / self.a

    def bar(self, x):
        y = self.foo(x)
        return y * self.b
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def foo(self, x):
        return x**2 / self.a

    def bar(self, x):
        y = self.foo(x)
        return y * self.b

bazz = Baz(1, 2)
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def foo(self, x):
        return x**2 / self.a

    def bar(self, x):
        y = self.foo(x)
        return y * self.b

bazz = Baz(1, 2)      bazz.bar(10) ??
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def foo(self, x):
        return x**2 / self.a

    def bar(self, x):
        y = self.foo(x)
        return y * self.b

bazz = Baz(1, 2)

bazz.bar(10) = self.foo(10) * self.b
```

Object-Oriented Programming (OOP)

```
class Baz(object):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def foo(self, x):
        return x**2 / self.a

  def bar(self, x):
        y = self.foo(x)
        return y * self.b

bazz = Baz(1, 2)

bazz.bar(10) = self.foo(10) * self.b
             = (10**2)/self.a * self.b = 200
```

The **Scikit-Learn** setup.

```
class ModelType(object):
```

The **Scikit-Learn** setup.

```
class ModelType(object):

    def __init__(self, some_arguments):
```

The **Scikit-Learn** setup.

```
class ModelType(object):

    def __init__(self, some_arguments):

    def fit(self, train_data, labels):
      actually does the parameter estimation
      doesn't return anything
```

The **Scikit-Learn** setup.

```
class ModelType(object):

    def __init__(self, some_arguments):

    def fit(self, train_data, labels):
      actually does the parameter estimation
      doesn't return anything

def decision_function(self, test_data):
  returns "confidence" (continuous value)
  for each class for each test data point
     shape of return:
        num_samples x num_categories
```

The **Scikit-Learn** setup.

```
class ModelType(object):

    def __init__(self, some_arguments):

    def fit(self, train_data, labels):
      actually does the parameter estimation
      doesn't return anything

  def decision_function(self, test_data):
    returns "confidence" (continuous value)
    for each class for each test data point
       shape of return:
          num_samples x num_categories

  def predict(self, test_data):
    returns actual discrete predictions
       shape of return:
          num_samples
```

Cross Validation

`Procedure:` (using the **Scikit-Learn** setup.)

```
splits = get_splits(split arguments)

For each split:
```

Cross Validation

Procedure: (using the **Scikit-Learn** setup.)

```
splits = get_splits(split arguments)

For each split:

  model = ModelType(args)
  model.fit(train_data, train_labels)
```

Cross Validation

Procedure: (using the **Scikit-Learn** setup.)

```
splits = get_splits(split arguments)

For each split:

  model = ModelType(args)
  model.fit(train_data, train_labels)

  prediction = model.predict(test_data)
  result = compare(prediction, test_labels)
```

Cross Validation

`Procedure:` (using the **Scikit-Learn** setup.)

```
splits = get_splits(split arguments)
```

Cross Validation

Procedure: (using the **Scikit-Learn** setup.)

```
splits = get_splits(split arguments)

For each split:

  model = ModelType(args)
  model.fit(train_data, train_labels)

  prediction = model.predict(test_data)
  result = compare(prediction, test_labels)

average results over splits
```

Cross Validation

Procedure: (using the **Scikit-Learn** setup.)

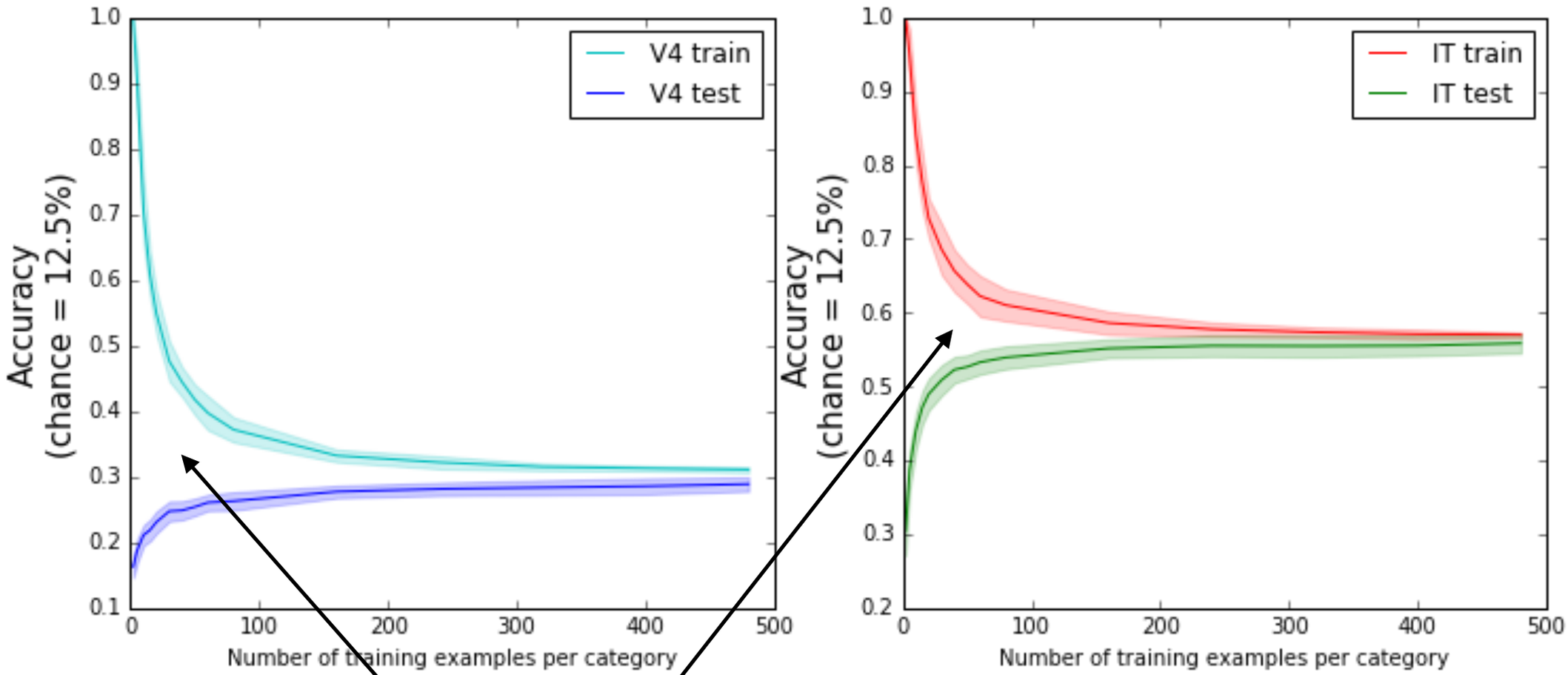
```
splits = get_splits(split arguments)

For each split:

  model = ModelType(args)
  model.fit(train_data, train_labels)

  prediction = model.predict(test_data)
  result = compare(prediction, test_labels)

average results over splits
```

creates a new model instance for each split

Cross Validation

[IPYNB: The Train-Test Procedure]

Cross Validation



OVERFITTING