Slide Type Sub-Slide

~~**Margarine, Popcorn, and Skynet's role in the Technological Singularity Event**~~

~~*aka Keanu Reeves rumored to be considering role in "A.I. Artificial Intelligence" prequel*~~

Soft Margins, the Kernel Trick, and Support Vector Machines

aka, Support Vector Classifiers

Scott Lee Schwartz, Ph.D.

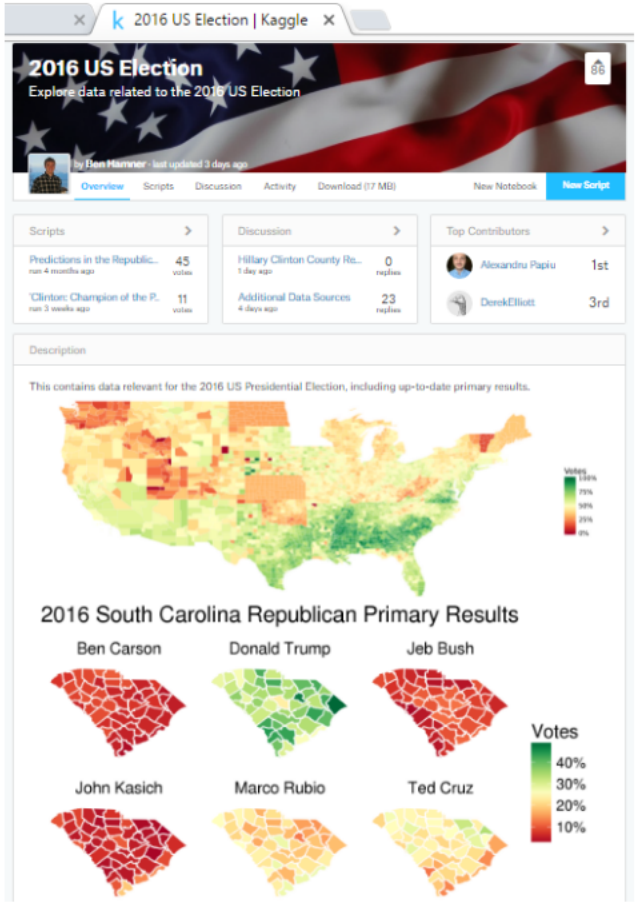
In [1]:

Slide Type

Slide Type Sub-Slide ▼

Happy (belated) Fourth of July!

- To celebrate, I have
  - a lot of red, white and blue plots
  - Donald Trump



Republican Primaries (Data available on Kaggle)

County level demographic data

- % < 18 years old
- % with a HS degree
- median household income
- population per square-mile
- and much, much more...

County level primary voting outcomes

- I characterize counties as *strongly* pro Donald (or not)
- in "one-versus-all" characterization

County level demographic data

- Suppose this is a subset of all county results -- those where primary voting has completed
- We might imagine predicting outcomes in future county primary elections based on currently available results

File Edit View Insert Cell Kernel Help Python 3

In [2]: Slide Type Slide

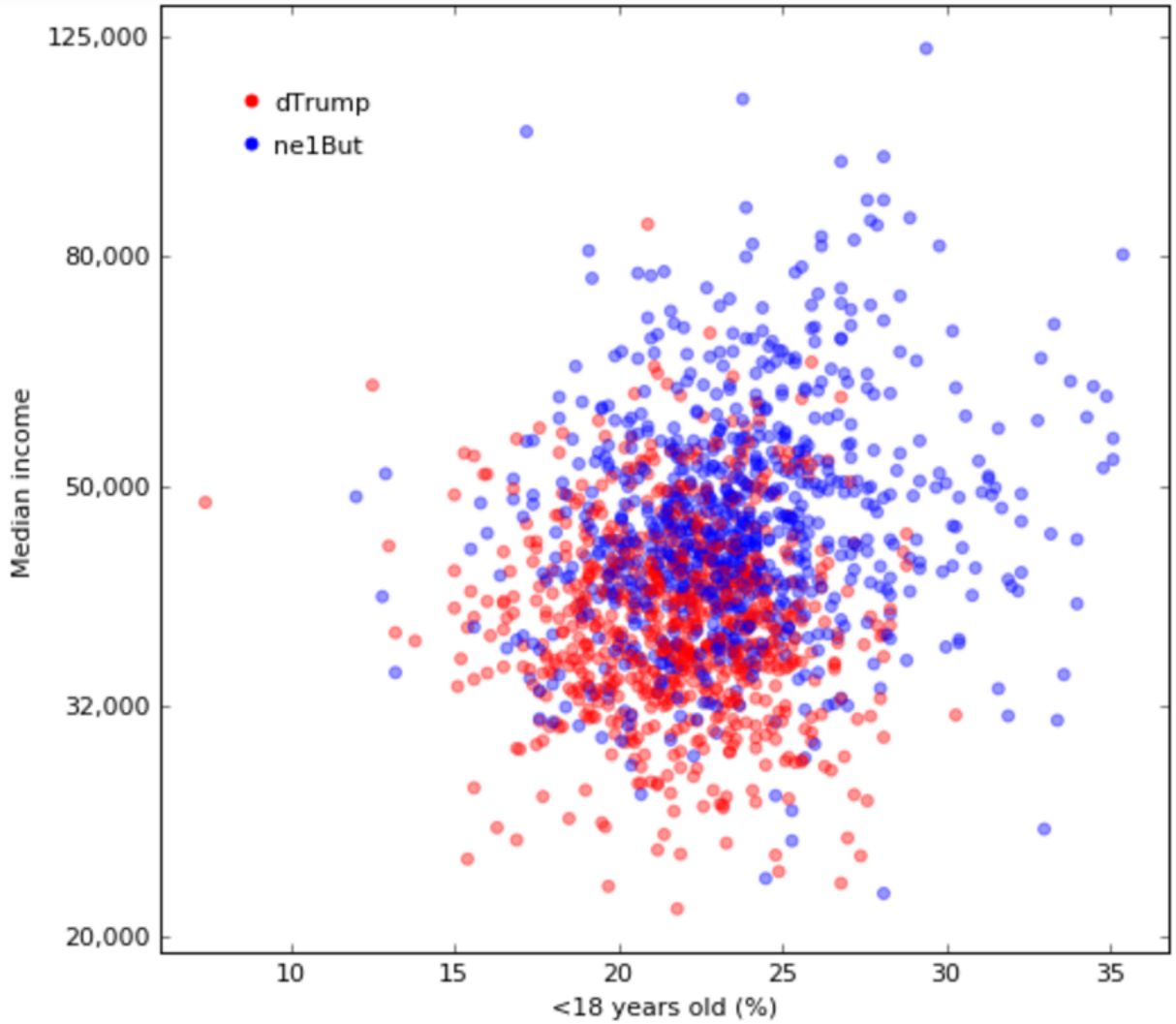
```
import pandas as pd
import numpy as np

rp=pd.read_csv("DTRprimaries.csv")
rp=rp.reindex(np.random.permutation(rp.index))
rp[:13]
```

Out[2]:

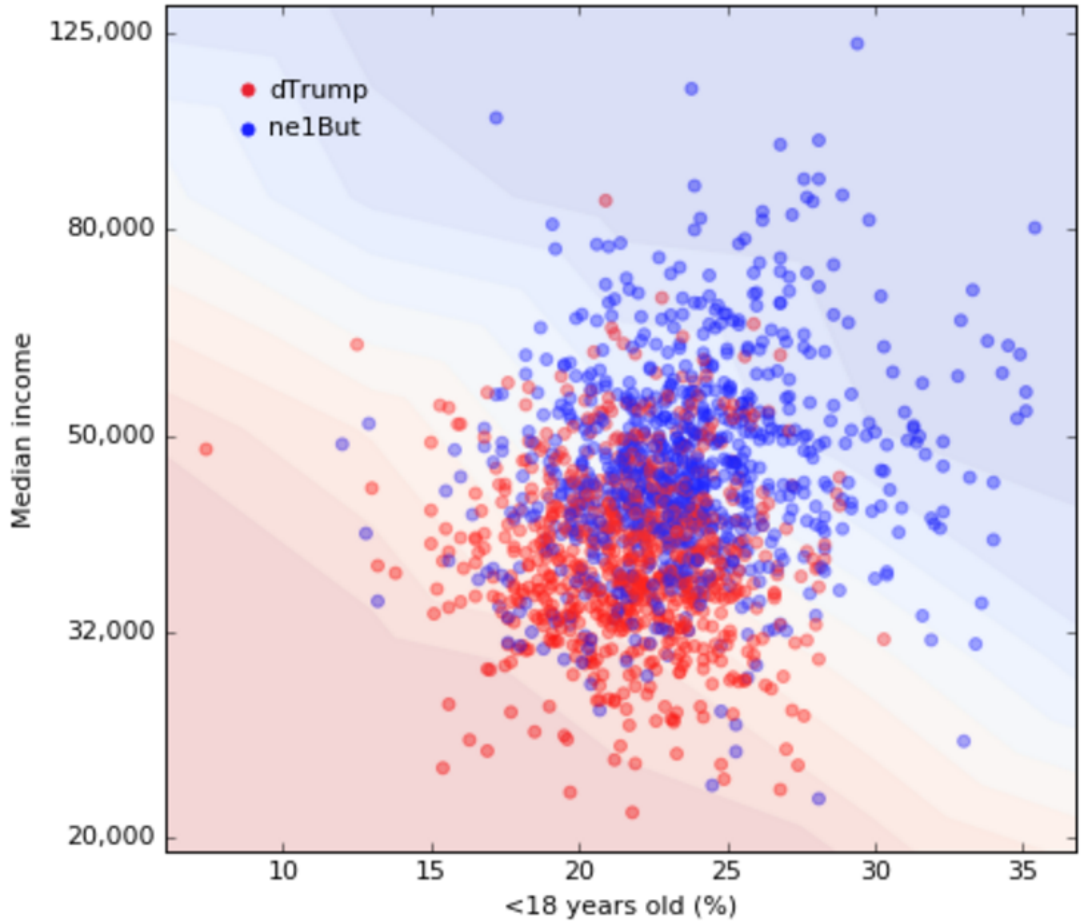
	DT	under18	withhs	medincome	ppm2
766	1	22.2	81.0	31132	8.4
146	0	23.8	82.6	42487	74.5
808	0	18.8	72.6	30768	30.4
1327	0	25.5	65.3	43438	0.3
1214	0	21.6	92.1	71607	0.7
957	1	24.2	84.2	41446	88.4
685	0	21.5	86.0	47964	53.1
220	1	23.6	82.3	47087	394.3
101	0	23.5	84.9	49004	28.3
1085	0	23.6	84.9	43765	84.9
846	0	17.1	85.0	36951	82.1
1310	0	21.9	74.1	45612	38.9
1189	1	23.3	73.1	34399	85.5

In [3]: Slide Type Fragment



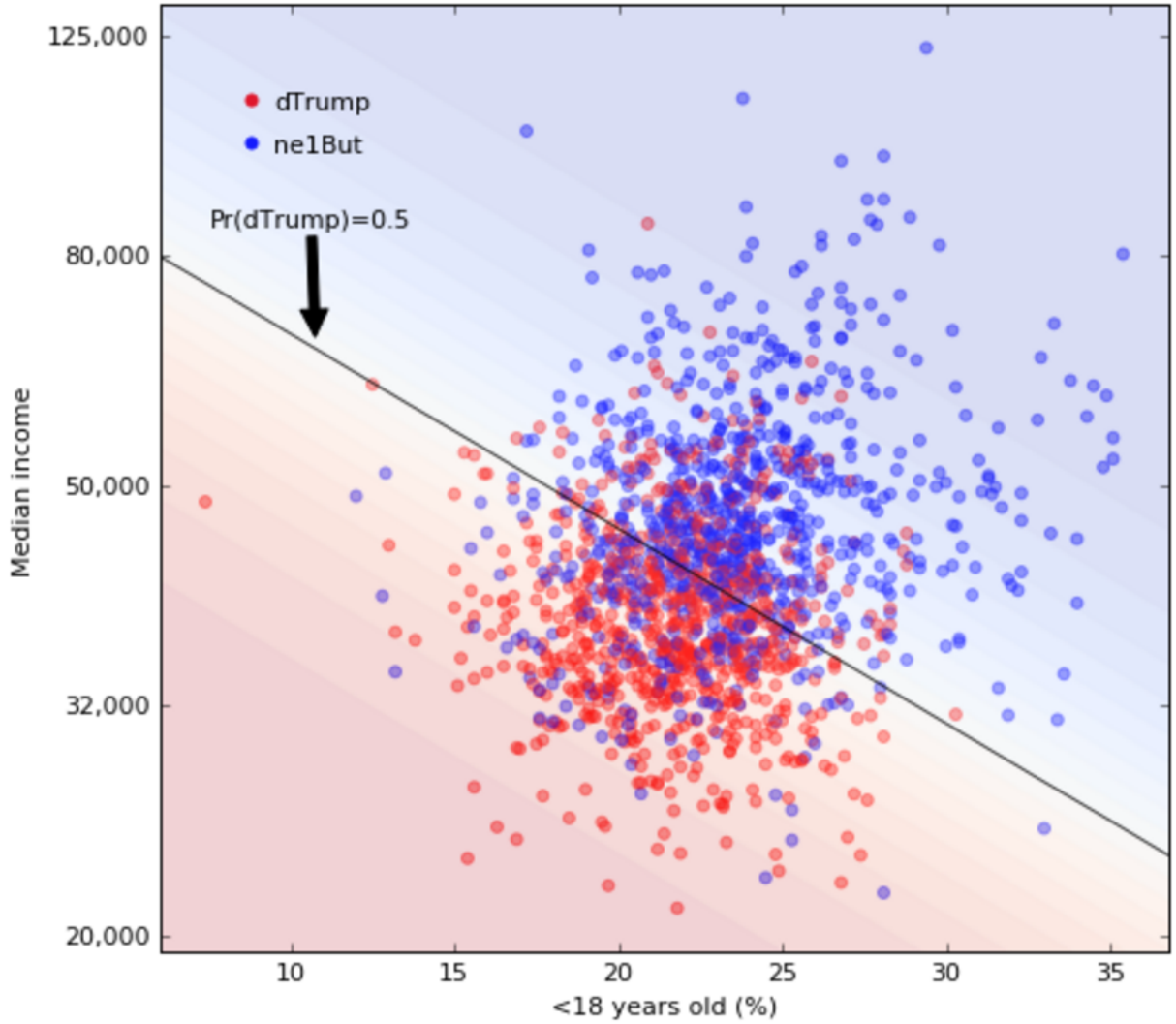
Slide Type Sub-Slide ▾

- Younger counties tend to *not* be so into Donald
- Wealthier counties tend to *not* be so into Donald



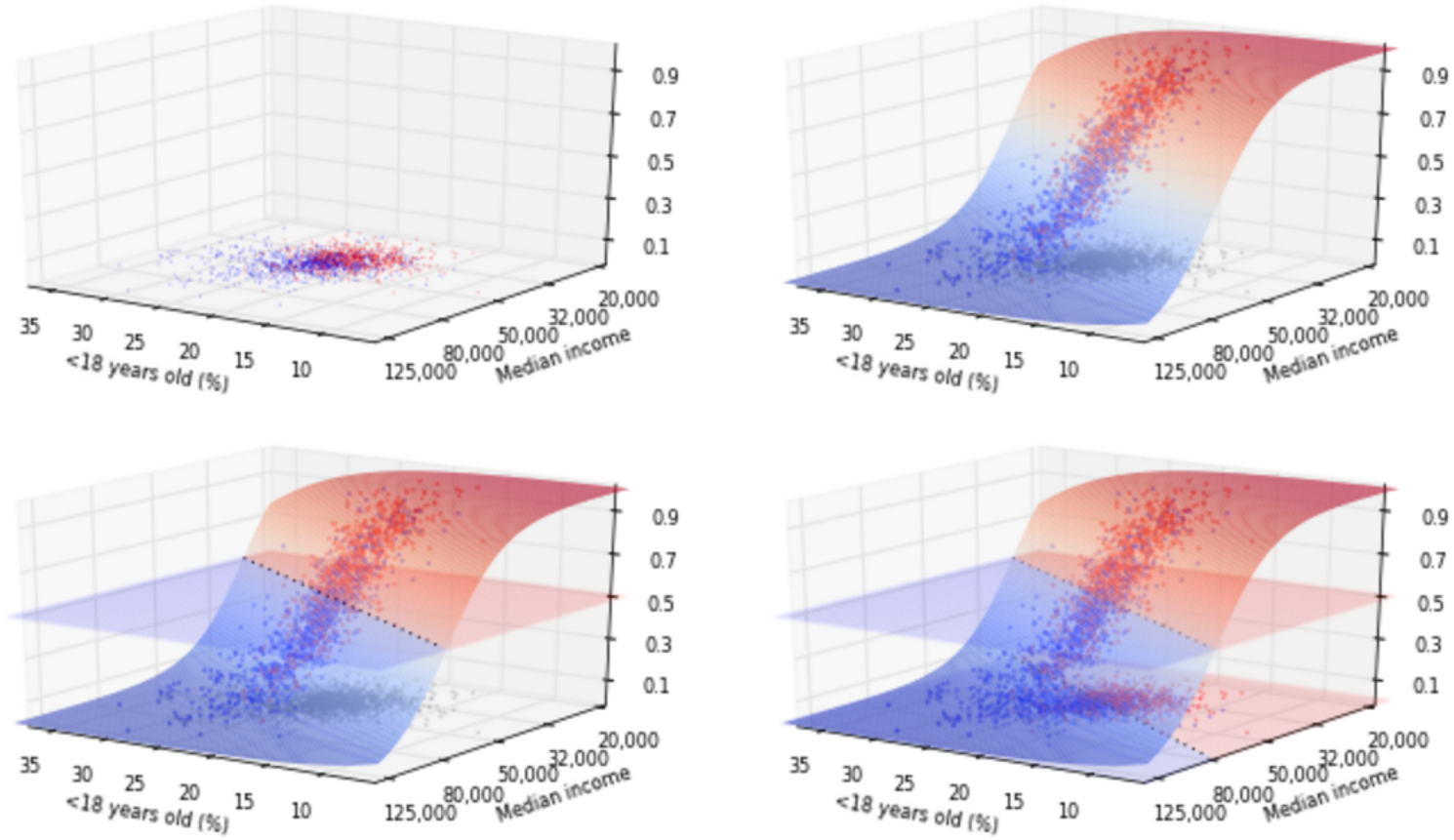
Slide Type Sub-Slide ▼

- Essentially, preference of the Donald *decreases* as we move towards the upper left (Northwest) of the plot
- While preference of the Donald *increases* as we move towards the upper left (Northwest) of the plot
- Let's explicity model these preferences...
  - We shall estimate the probability (based on the data) that a county goes Donald given Age and Income
  - We shall do this with the vanilla logistic regression (linear) model "DT~Age+Income"



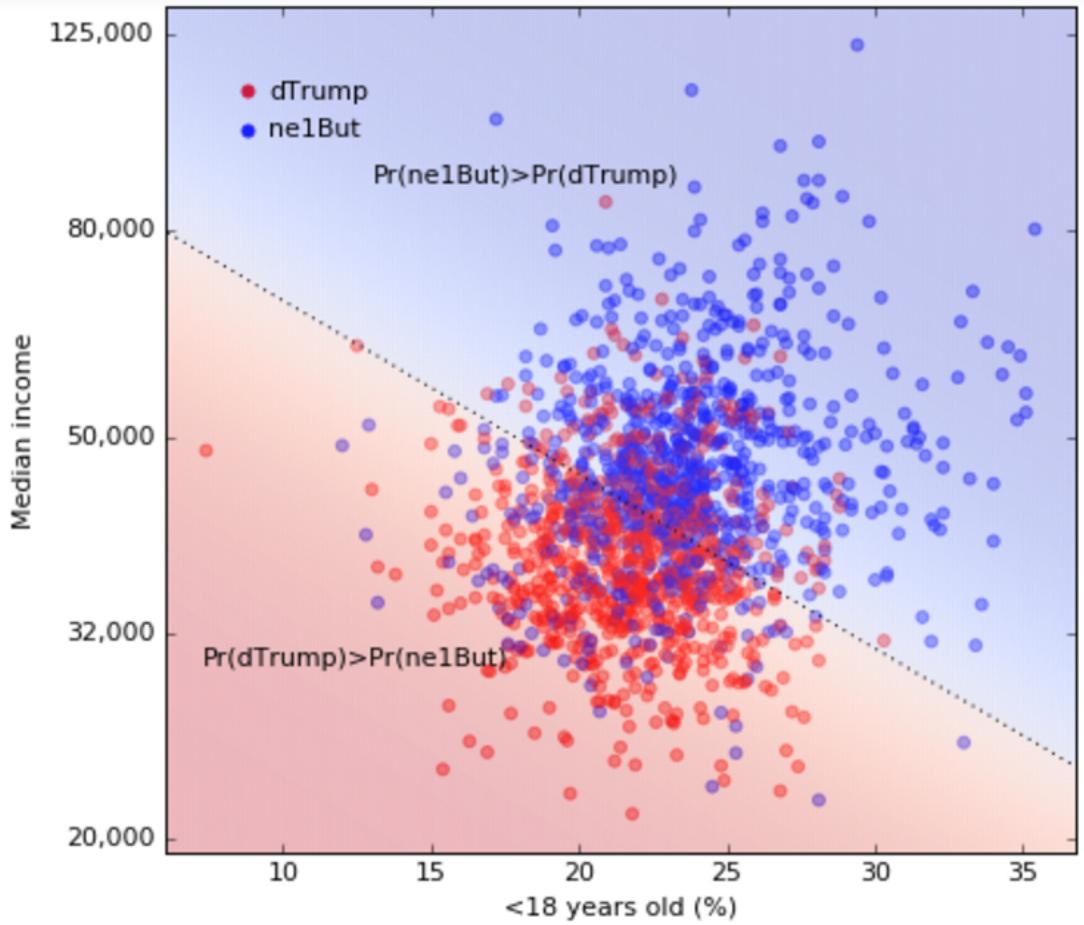
Slide Type Sub-Slide ▼

- The logistic regression model ("DT~Age+Income") estimates a probability gradient as a function of Age and Income
- Here we can see all the combinations of Age and Income which result in a 50/50 chance the county goes Donald



Slide Type Sub-Slide ▼

- (1) The top left figure shows our the county level Age and Income data
- (2) The top right figure shows the probability (surface) as a function of Age and Income
  - We estimate the probability a county goes Donald with the "DT~Age+Income" logistic regression model
    - Note the sigmoidal shape of the probability curve -- this is a property of logistic regression
- (3) The bottom left plot shows where the curve hits 50% (i.e., where it intersects with the z+0y+0x=0.5 plane)
- (4) The bottom right plot projects this intersection down into the original space (the z+0y+0x=0.0 plane)



Slide Type Sub-Slide ▼

- So, if our objective is to predict future voting outcomes based on these data...
- then we can theoretically maximize our accuracy by guessing the outcome with the highest predicted probability

So estimating probabilities is a way to make a *classification* prediction decision

- Let's explore this further in a much simpler, smaller case

Slide Type Sub-Slide

- So, if our objective is to predict future voting outcomes based on these data...
- then we can theoretically maximize our accuracy by guessing the outcome with the highest predicted probability

So estimating probabilities is a way to make a *classification* prediction decision

- Let's explore this further in a much simpler, smaller case

In [12]:

Slide Type

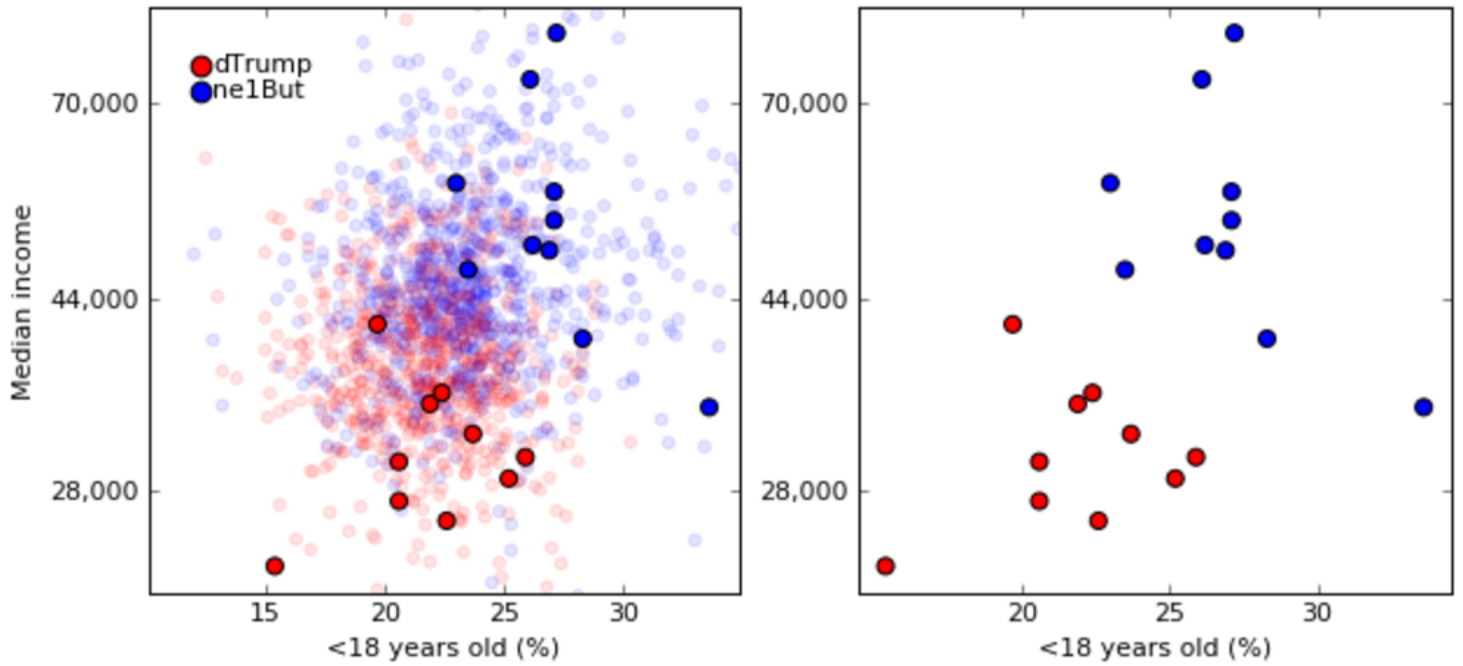
```
rp_s=pd.read_csv("DTRprimaries_small.csv")
pd.concat([rp_s[:10],rp_s[10:].reset_index(drop=True)],axis=1)
```

Out[12]:

	DT	under18	medincome	DT	under18	medincome
0	1	20.6	27353	0	26.2	50112
1	1	21.9	34419	0	27.2	82762
2	1	25.9	30345	0	26.9	49481
3	1	25.2	28853	0	26.1	74155
4	1	22.6	26104	0	28.3	40157
5	1	23.7	32054	0	27.1	53137
6	1	19.7	41550	0	33.6	34146
7	1	20.6	30023	0	23.5	47277
8	1	15.4	23451	0	27.1	56853
9	1	22.4	35329	0	23.0	58025

Out[13]:

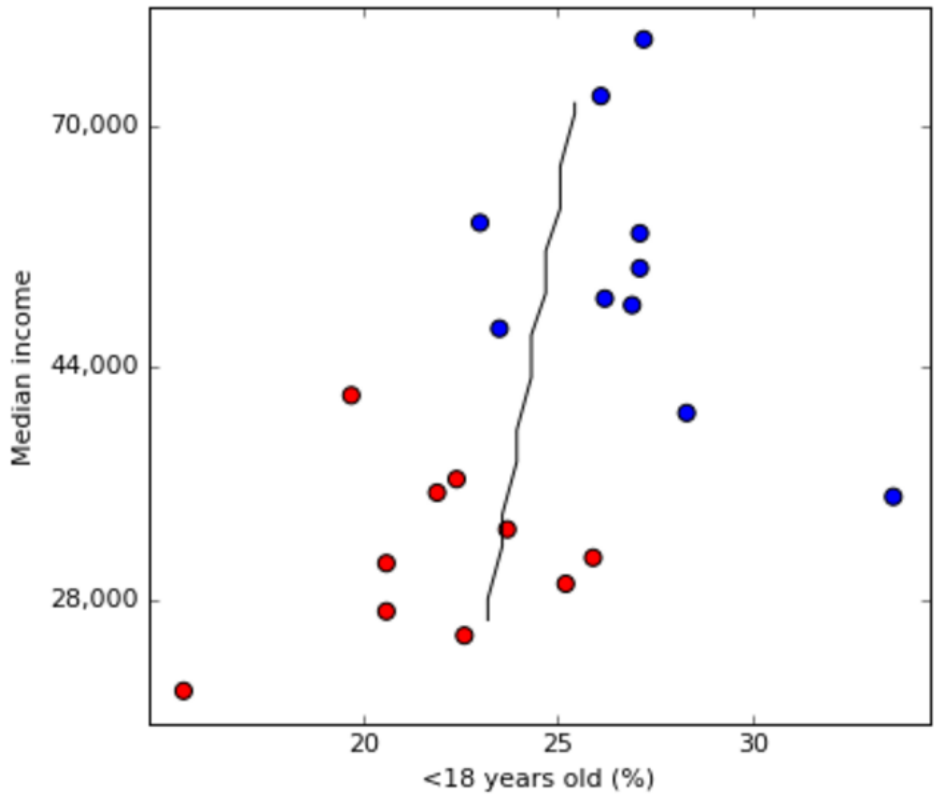
	DT	under18	medincome	DT	under18	medincome	DT	under18	medincome	DT	under18	medincome
0	1	20.6	27353	1	23.7	32054	0	26.2	50112	0	27.1	53137
1	1	21.9	34419	1	19.7	41550	0	27.2	82762	0	33.6	34146
2	1	25.9	30345	1	20.6	30023	0	26.9	49481	0	23.5	47277
3	1	25.2	28853	1	15.4	23451	0	26.1	74155	0	27.1	56853
4	1	22.6	26104	1	22.4	35329	0	28.3	40157	0	23.0	58025



Slide Type Sub-Slide ▼

- So we select 10 counties that were pro Donald and 10 that weren't and just consider these
- Importantly, we have selected 10 counties that are clearly "separable"

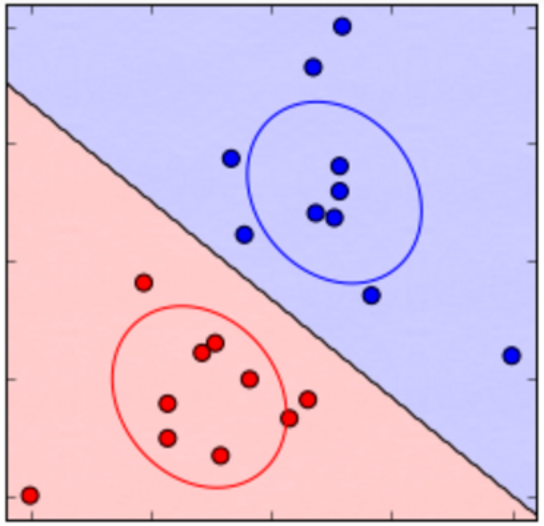
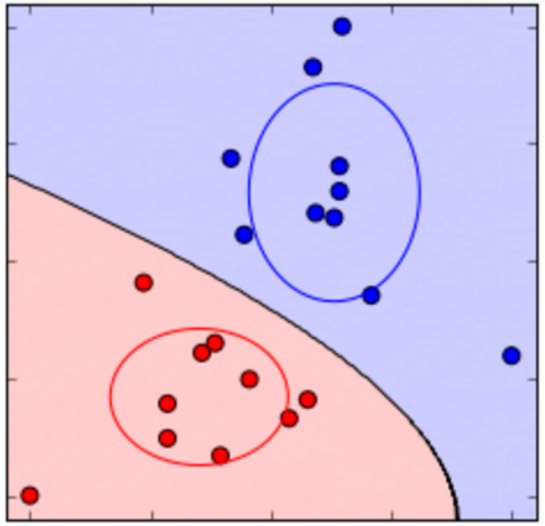
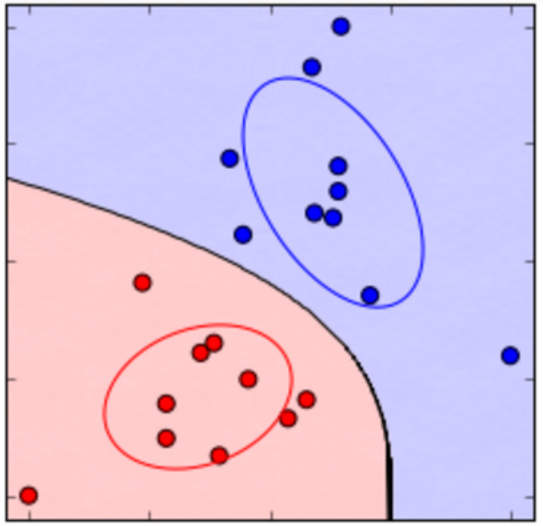
```
c:\users\scott\anaconda3\lib\site-packages\sklearn\linear_model\sag.py:267: ConvergenceWarning: The ma
x_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```



Slide Type Sub-Slide ▼

- If we start the same way we run into the unexpected problem that the the problem is *too easy* for logistic regression
  - (1) there are infinitely many solutions, and (2) these coefficient estimates that approach infinity

We're just want a decision boundary to predict red or blue here... this should be easy

Slide Type Sub-Slide ▼

Rather than logistic regression, I now use discriminant analysis

- Regression estimates a distribution of variable conditional on covariates
- Discriminant analysis estimates the joint distribution of all variables
  - **and then uses Bayes' Theorem to estimate the conditional distribution of the outcome**

One might make several parametric modeling assumptions in doing this

- E.g., one might specify **normality** and the structure of **covariance matrix** therein
  - A common covariance matrix is called **LDA** (*Linear Discriminant Analysis*)
  - Unique, but diagonal covariance matricies (i.e., independent covariates) is called **Naive Bayes**
  - Unique, unstructured covariance matricies is called **QDA** (*Quadratic Discriminant Analysis*)

These all just produce a decision boundary with varying degrees of flexibility

These all just produce a decision boundary with varying degrees of flexibility

- And discriminant analysis is just one class of many available for doing this...

In [17]:

Slide Type

```
from sklearn.neighbors import KNeighborsClassifier

#from sklearn.neural_network import MLPClassifier

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import SGDClassifier
```

Slide Type Sub-Slide

- So, as a statistician, my first thought when approaching a classification problem is *"Logistic Regression"*
- Alternatively, my engineering friends who do "signal processing", would instead think *"Discriminant Analysis"*
- And other analysts might think *"Nearest Neighbors"*, *"Neural Networks"*, or *"Tree-based methods"*, etc.

And rightly so: all have seen success in various applications

These days, however, there is a new tool that's very hot and popular

- A computer scientist, from the machine learning community, would therefore probably first think *"Support Vector Machine..."*

- And discriminant analysis is just one class of many available for doing this...

In [17]:

Slide Type ▼

```
from sklearn.neighbors import KNeighborsClassifier

#from sklearn.neural_network import MLPClassifier

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import SGDClassifier
```

Slide Type Sub-Slide ▼

- So, as a statistician, my first thought when approaching a classification problem is *"Logistic Regression"*
- Alternatively, my engineering friends who do "signal processing", would instead think *"Discriminant Analysis"*
- And other analysts might think *"Nearest Neighbors"*, *"Neural Networks"*, or *"Tree-based methods"*, etc.

And rightly so: all have seen success in various applications

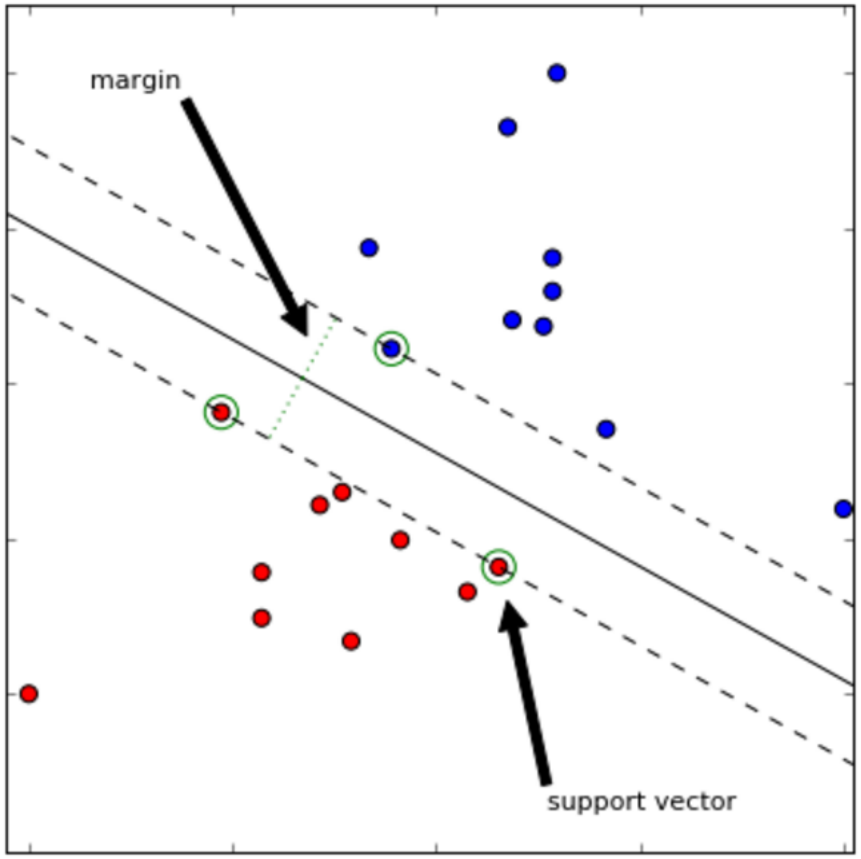
These days, however, there is a new tool that's very hot and popular

- A computer scientist, from the machine learning community, would therefore probably first think *"Support Vector Machine..."*

In [18]:

Slide Type ▼

```
from sklearn.svm import SVC
```



Slide Type Sub-Slide ▾

Support Vector Classifiers

- find a "separating hyperplane" that classifies the data
- Initially, we'll say "perfectly" separating the data, but this will quickly be relaxed via the "soft margin"
- There are many such separating hyperplanes, so which shall we choose?
- We'll take the hyperplane which "maximizes the margin" such that no points violate the margin

Slide Type Sub-Slide

- A hyperplane in $\mathbb{R}^P$ is defined as

$$\mathbf{w}'\mathbf{x} + b = 0, x \in \mathbb{R}^P$$

E.g., $\mathbf{w} = (1, -1)$ *and* $b = 0$ *defines the 45 degree line in* $\mathbb{R}^2$ *(the so called "y = x" line)*

- $\mathbf{w}$ is perpendicular to $\mathbf{w}'\mathbf{x} + b = 0$

Suppose $\mathbf{w}'\mathbf{x}^* + b = 0$ and $\mathbf{w}'\mathbf{x}^{**} + b = 0$	[$\mathbf{x}^*$ *and* $\mathbf{x}^{**}$ *lie on the hyperplane*]
Then $\mathbf{w}'(\mathbf{x}^* - \mathbf{x}^{**}) + b = 0$	[$\mathbf{w}$ *is perpendicular to* $(\mathbf{x}^* - \mathbf{x}^{**})$]
And so $\mathbf{x}$ must be perpendicular to $\mathbf{x}'\mathbf{x} + b = 0$	[*since* $(\mathbf{x}^* - \mathbf{x}^{**})$ *is parallel* $\mathbf{w}'\mathbf{x} + b = 0$]

- Since $\mathbf{w}'\mathbf{x} + b = 0$ and $c(\mathbf{w}'\mathbf{x} + b) = 0$ define the same hyperplane for all $c \in \mathbb{R}$
- for a given data set $\{\mathbf{x}_n : n = 1, \cdots, N\}$, we select $\mathbf{w}$ such that

$$\min_{n=1,\cdots,N} |\mathbf{w}'\mathbf{x}_n + b| = 1$$

- And then for $|\mathbf{w}'\mathbf{x}_s + b| = 1$ and $\mathbf{w}'\mathbf{x}^* + b = 0$, we "maximize the margin":

$$\max_{\mathbf{w}} \frac{\mathbf{w}}{||\mathbf{w}||}(\mathbf{x}_s - \mathbf{x}^*) = \max_{\mathbf{w}} \frac{1}{||\mathbf{w}||}$$

where $||\mathbf{w}|| = \sqrt{\mathbf{w}'\mathbf{w}}$ *and* $\frac{\mathbf{w}}{||\mathbf{w}||}\mathbf{x}$ *is the projection of x onto w*

So how is this "maximizing" to find w done?

- Let $y_n \in \{-1, 1\}$ such that $y_n(\mathbf{w}'\mathbf{x}_n + b) \geq 1$, and optimize

$$\min_w \frac{\mathbf{w}'\mathbf{w}}{2} : (y_n(\mathbf{w}'\mathbf{x}_n + b) - 1) \geq 0 \qquad \textit{since } \min_{\mathbf{w}} \frac{\mathbf{w}'\mathbf{w}}{2} \textit{ is equivalent to } \max_{\mathbf{w}} \frac{1}{\|\mathbf{w}\|} = \max_{\mathbf{w}} \frac{1}{\sqrt{\mathbf{w}'\mathbf{w}}}$$

$$\textit{and } y_n(\mathbf{w}'\mathbf{x}_n + b) = 1 \textit{ for some } n \textit{ since } \mathbf{w}'\mathbf{w} \textit{ is being minimized}$$

- Which is equivalent to optimizing the Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{\mathbf{w}'\mathbf{w}}{2} - \sum_{n=1}^{N} \lambda_n (y_n(\mathbf{w}'\mathbf{x}_n + b) - 1) \qquad \lambda_n \geq 0$$

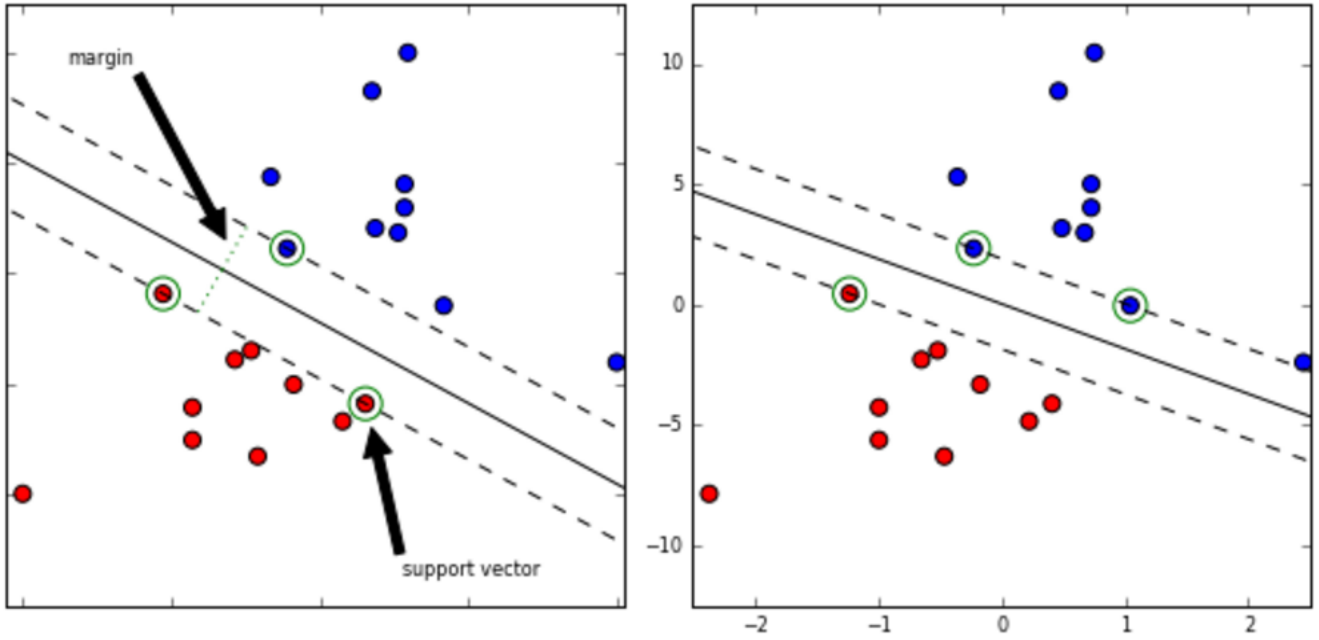
$$= \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \lambda_m y_m \mathbf{x}_m' \mathbf{x}_n y_n \lambda_n - b \sum_{n=1}^{N} \lambda_n y_n$$

$$\longrightarrow = \mathbf{1}'\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda} \quad (-\mathbf{I}\boldsymbol{\lambda} \leq \mathbf{0}, -\mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{y}'\boldsymbol{\lambda} \leq 0) \qquad Q_{mn} = y_m \mathbf{x}_m' \mathbf{x}_n y_n$$

$$\mathcal{L}_{\mathbf{w}}(\mathbf{w}, b, \boldsymbol{\lambda}) = \mathbf{w} - \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \qquad \mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n$$

$$\mathcal{L}_b(\mathbf{w}, b, \boldsymbol{\lambda}) = \sum_{n=1}^{N} \lambda_n y_n \qquad 0 = \sum_{n=1}^{N} \lambda_n y_n$$

- where the indicated expression can be optimized via quadratic programming



Slide Type Sub-Slide

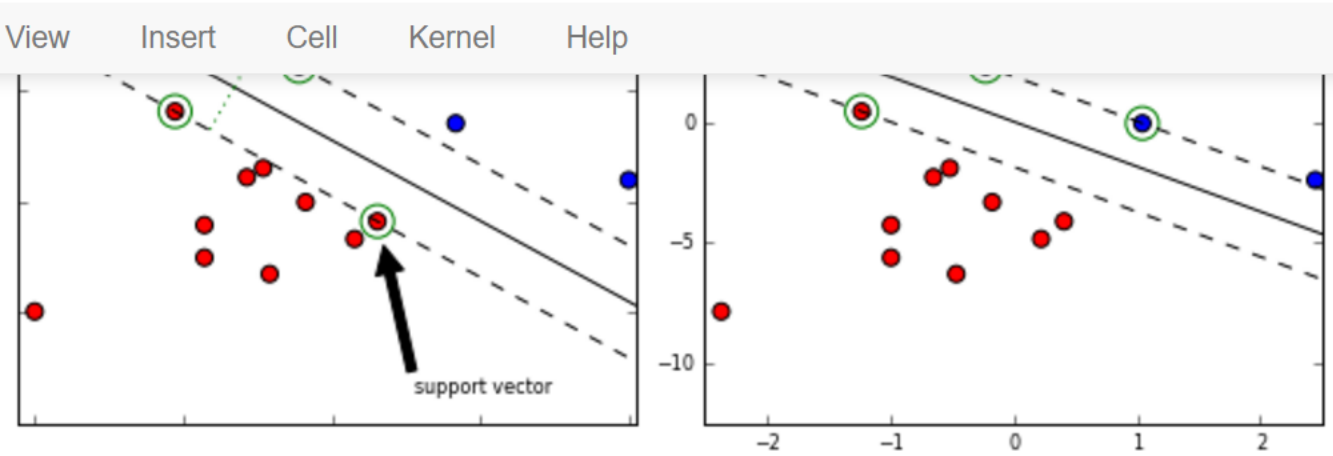
Given $\{(y_n, \mathbf{x}_n) : n = 1, \cdots N\}$, optimization via quadratic programming provides $\boldsymbol{\lambda}$ from which we calculate

$$\mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n$$

- Interestingly, $\boldsymbol{\lambda}$ is sparse -- i.e., many $\{\lambda_n : n = 1, \cdots N\}$ are zero

  $\{\mathbf{x}_s : \lambda_s > 0, s = 1, \cdots N\}$ are the so called **support vectors**

- $\mathbf{w}'\mathbf{x} + b = 0$ defines the separating hyperplane with the largest margin $\left(\frac{1}{\|\mathbf{w}\|}\right)$
  - b is calculated from the support vectors $\{\mathbf{x}_s\}$ and $\mathbf{w}$, subject to the constraint that $|\mathbf{w}'\mathbf{x}_s + b| = 1$



Slide Type Sub-Slide ▾

Given $\{(y_n, \mathbf{x}_n) : n = 1, \cdots N\}$, optimization via quadratic programming provides $\boldsymbol{\lambda}$ from which we calculate

$$\mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n$$

- Interestingly, $\boldsymbol{\lambda}$ is sparse -- i.e., many $\{\lambda_n : n = 1, \cdots N\}$ are zero

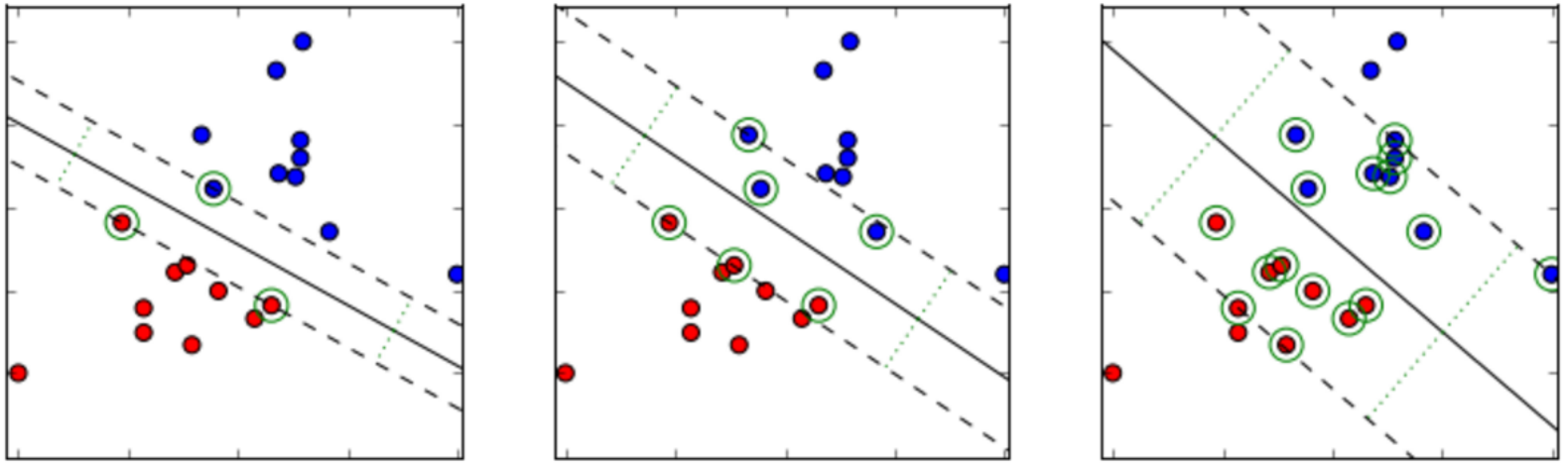
  $\{\mathbf{x}_s : \lambda_s > 0, s = 1, \cdots N\}$ are the so called **support vectors**

- $\mathbf{w}'\mathbf{x} + b = 0$ defines the separating hyperplane with the largest margin $\left(\frac{1}{\|\mathbf{w}\|}\right)$
  - b is calculated from the support vectors $\{\mathbf{x}_s\}$ and $\mathbf{w}$, subject to the constraint that $|\mathbf{w}'\mathbf{x}_s + b| = 1$

Slide Type Sub-Slide ▾

Note that the scale of each dimension of x matters since $\|\mathbf{w}\| = \sqrt{\sum_{m=1}^{P}(\lambda_1 y_1 x_{1m})^2 + \cdots + (\lambda_n y_n x_{nm})^2}$

is not scale aware and equivalently penalizes each dimension of $\mathbf{x}_n$ ***with respect*** $\mathbb{R}^1$



Slide Type Sub-Slide

- Requiring data to be perfectly (linearly) separable *seems* like a prohibative restriction
- We therefore allow each point $\mathbf{x}_n$ to violate the margin by some amount ϵ_n, i.e.

$$y_n(\mathbf{w}'\mathbf{x}_n + b) \geq 1 - \epsilon_n, \quad \epsilon_n \geq 0$$

- and (for $C > 0$) instead optimize

$$\min_{\mathbf{w},\boldsymbol{\epsilon}} \frac{\mathbf{w}'\mathbf{w}}{2} + C\sum_{n=1}^{N} \epsilon_n: \quad y_n(\mathbf{w}'\mathbf{x}_n + b) \geq 1 - \epsilon_n, \quad \epsilon_n \geq 0$$

- which results in a nearly identical quadratic programming problem as before but with an added constraint for C

$$\mathbf{1}'\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda} \quad (-\mathbf{I}\boldsymbol{\lambda} \leq \mathbf{0}, -\mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{I}\boldsymbol{\lambda} \leq \mathbf{C})$$

Slide Type Sub-Slide

Slide Type Sub-Slide

- Requiring data to be perfectly (linearly) separable *seems* like a prohibative restriction
- We therefore allow each point $\mathbf{x}_n$ to violate the margin by some amount ϵ_n, i.e.

$$y_n(\mathbf{w}'\mathbf{x}_n + b) \geq 1 - \epsilon_n, \ \epsilon_n \geq 0$$

- and (for $C > 0$) instead optimize

$$\min_{\mathbf{w},\boldsymbol{\epsilon}} \frac{\mathbf{w}'\mathbf{w}}{2} + C\sum_{n=1}^{N} \epsilon_n : \quad y_n(\mathbf{w}'\mathbf{x}_n + b) \geq 1 - \epsilon_n, \ \epsilon_n \geq 0$$

- which results in a nearly identical quadratic programming problem as before but with an added constraint for C

$$\mathbf{1}'\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda} \quad (-\mathbf{I}\boldsymbol{\lambda} \leq \mathbf{0}, -\mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{I}\boldsymbol{\lambda} \leq \mathbf{C})$$

Slide Type Sub-Slide

This is the *"soft margin" support vector classifier* with penalty C

- *The support vectors are now the set of points which touch or pass the margin* $\{\mathbf{x}_s : |\mathbf{w}'\mathbf{x} + b| \leq 1, s = 1, \cdots N\}$
- C influences the width of margin, i.e., number of support vectors influencing the separating hyperplane
  - Large C heavily penalizes margin violations, i.e. favors smaller margins and fewer support vectors
  - Small C tolerates increased violation, i.e., favors wider margins more support vectors
  - Data on the "wrong side" of the separating plane are necessarily support vectors, so increasing the margin means using more points on the "right side" of the separating plane to inform the separating plane
- Reducing C performs *"regularization"* by shrinking estimation back towards the global estimate based on all the data

Slide Type Sub-Slide

- The "soft margin" support vector classifier with

$$y_n(\mathbf{w}'\mathbf{x}_n + b - 1 + \epsilon_n) \geq 0, \epsilon_n \geq 0, \text{for } n = 1, \cdots, N \text{ and penalty } C$$

- optimizes the *new* Lagrangian

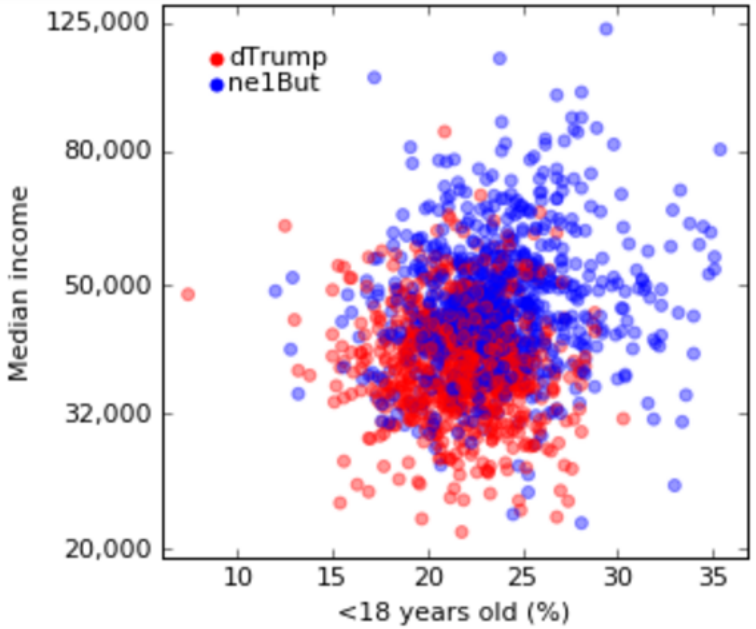
$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}, \boldsymbol{\epsilon}, \boldsymbol{\eta}) = \frac{\mathbf{w}'\mathbf{w}}{2} + C\sum_{n=1}^{N}\epsilon_n - \sum_{n=1}^{N}\lambda_n(y_n(\mathbf{w}'\mathbf{x}_n + b) - 1 + \epsilon_n) - \sum_{n=1}^{N}\eta_n\epsilon_n \qquad \lambda_n \geq 0, \eta_n \geq 0$$

$$\begin{aligned}
&= \sum_{n=1}^{N}\lambda_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\lambda_m y_m \mathbf{x}_m'\mathbf{x}_n y_n \lambda_n - b\sum_{n=1}^{N}\lambda_n y_n + \sum_{n=1}^{N}(C - \lambda_n - \eta_n)\epsilon_n \\
&= \sum_{n=1}^{N}\lambda_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\lambda_m y_m \mathbf{x}_m'\mathbf{x}_n y_n \lambda_n \\
&= \mathbf{1}'\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda} \quad (-\mathbf{I}\boldsymbol{\lambda} \leq \mathbf{0}, -\mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{y}'\boldsymbol{\lambda} \leq 0, \mathbf{I}\boldsymbol{\lambda} \leq \mathbf{C}) \qquad Q_{mn} = y_m\mathbf{x}_m'\mathbf{x}_n y_n
\end{aligned}$$

$$\mathcal{L}_{\mathbf{w}}(\mathbf{w}, b, \boldsymbol{\lambda}, \boldsymbol{\epsilon}, \boldsymbol{\eta}) = \mathbf{w} - \sum_{n=1}^{N}\lambda_n y_n \mathbf{x}_n \qquad \mathbf{w} = \sum_{n=1}^{N}\lambda_n y_n \mathbf{x}_n$$

$$\mathcal{L}_b(\mathbf{w}, b, \boldsymbol{\lambda}, \boldsymbol{\epsilon}, \boldsymbol{\eta}) = \sum_{n=1}^{N}\lambda_n y_n \qquad 0 = \sum_{n=1}^{N}\lambda_n y_n$$

$$\mathcal{L}_{\epsilon_n}(\mathbf{w}, b, \boldsymbol{\lambda}, \boldsymbol{\epsilon}, \boldsymbol{\eta}) = C - \lambda_n - \eta_n \qquad 0 = C - \lambda_n - \eta_n, \quad \lambda_n \leq C$$

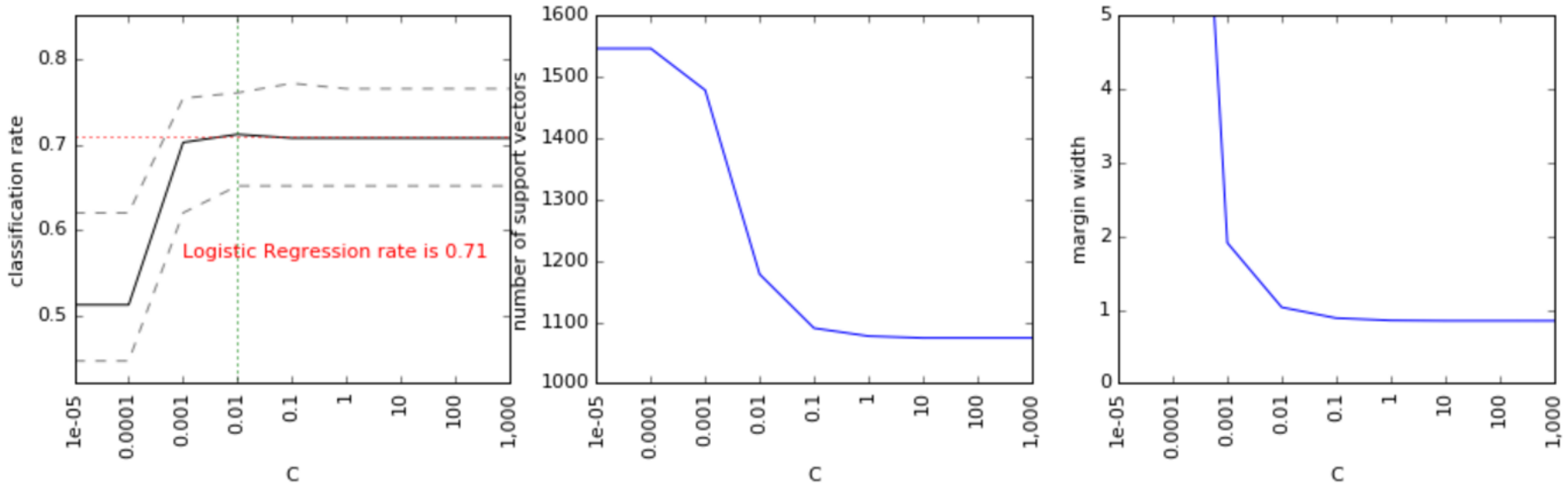


Slide Type Sub-Slide ▼

- Okay, back to the data
  - remember, we're trying to predict if a county will go Donald based on characteristics of it's Age and Income
- The data is not (linearly) separable
- We therefore (seem to) have to use the soft margin support vector classifier for this data
- This requires a choice for C
- Remember, C controls the tradeoff of the margin size $\frac{1}{||\mathbf{w}||}$ and the cumulative violations of the margin $\sum_{n=1}^{N} \epsilon_n$ in

$$\min_{\mathbf{w},\boldsymbol{\epsilon}} \frac{\mathbf{w}'\mathbf{w}}{2} + C\sum_{n=1}^{N} \epsilon_n$$

- Large C heavily penalizes margin violations (smaller margins) while small C tolerates more violation (wider margins)
- **We will choose C by cross-validation**



Slide Type Sub-Slide ▼

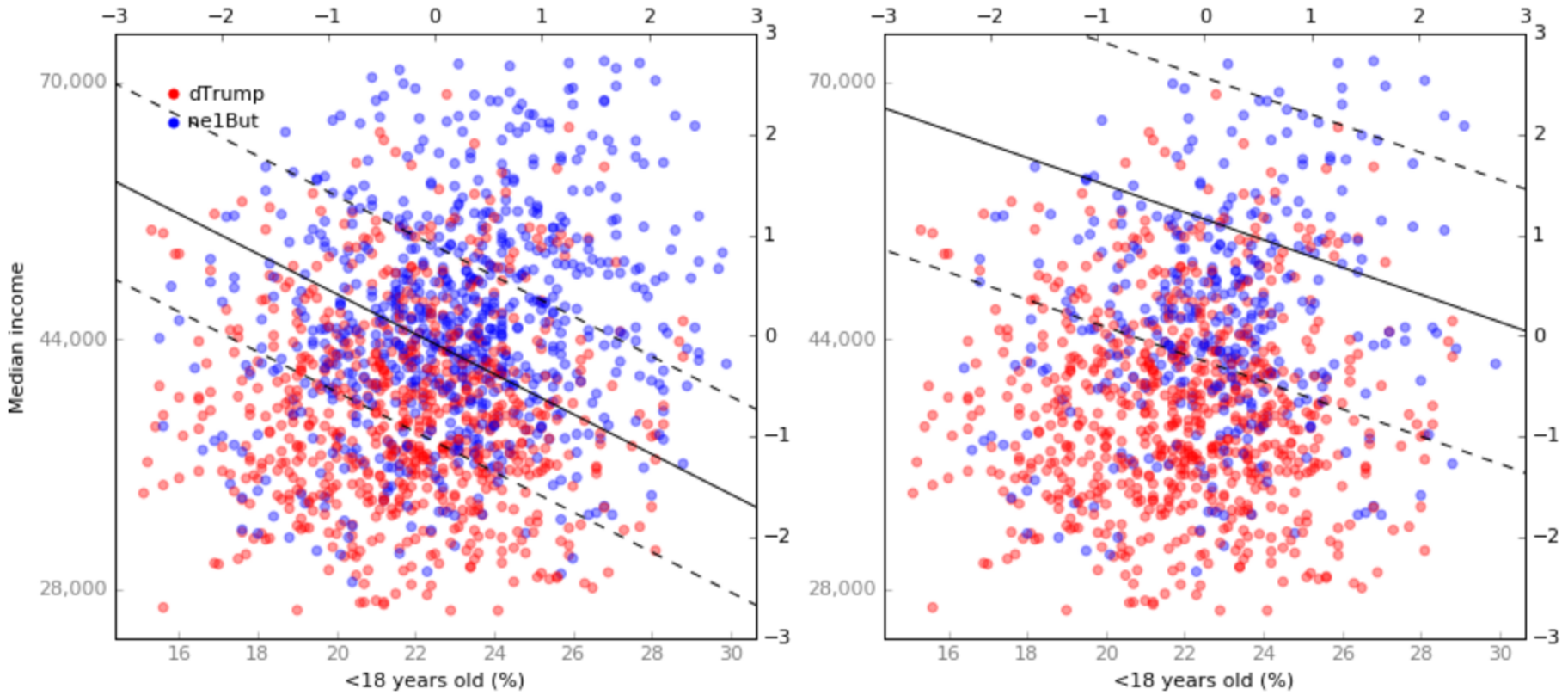
K-folds cross-validation

- Partitions the data into K folds
- Scores the prediction of each of the K folds based on the fit of the other K-1 folds.
- This gives K "out of sample" estimates of prediction accuracy, i.e., model estimation variance (overfitting)

K-folds parameter tuning

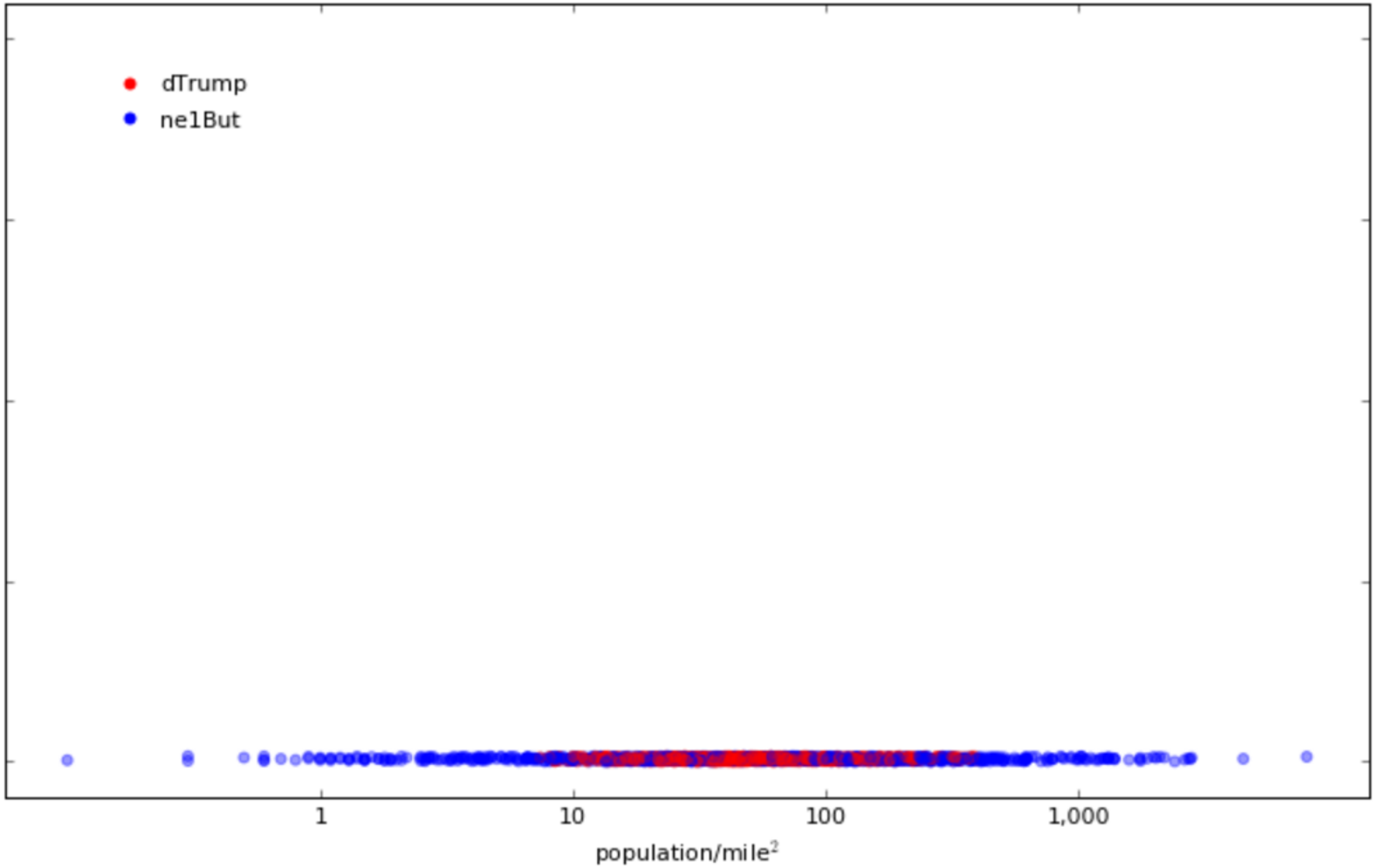
- Comparing K-folds prediction accuracy at different levels of C can be used to select a C
- We choose C with strongest "out of sample" prediction accuracy (and least amount of observed overfitting)
- It is still optimized with respect to the data at hand...

- ***Smaller C means (a) increased margins, thus (b) more support vectors, thus (c) more regularization***



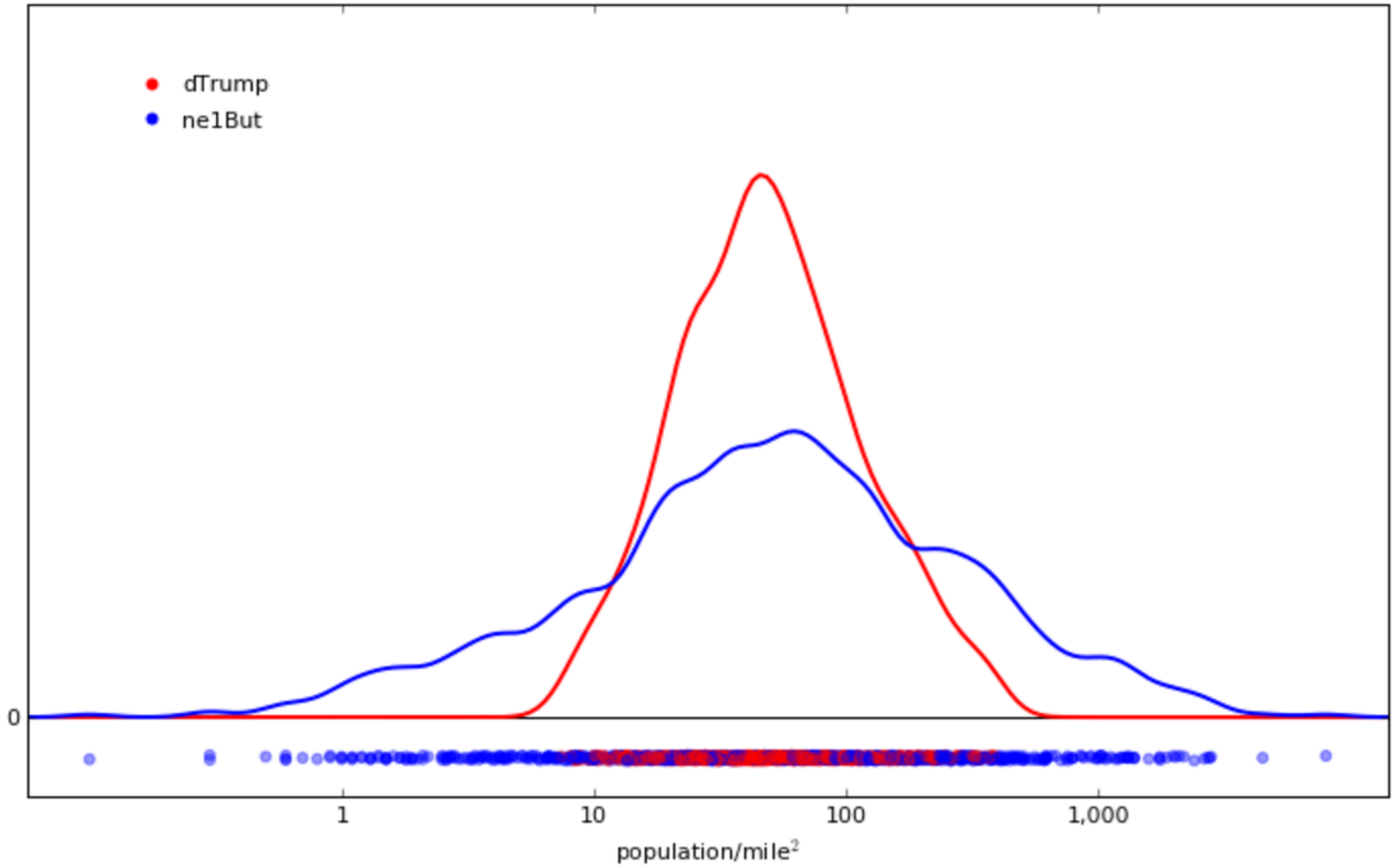
Slide Type Sub-Slide ▼

- Here is the SVC fit on the full data set
  - blue points on the bottom and red points on the top are support vectors
  - all points in the margin are support vector
  - we acheived an in sample classification rate of 71% -- same as logistic regression
- If the number of data points in each class is different the SVC will prefer the larger class
  - (since it is trying to make fewer mistakes overall)
  - the data can be weighted to balance the penalties between class and hence "normalize" class size



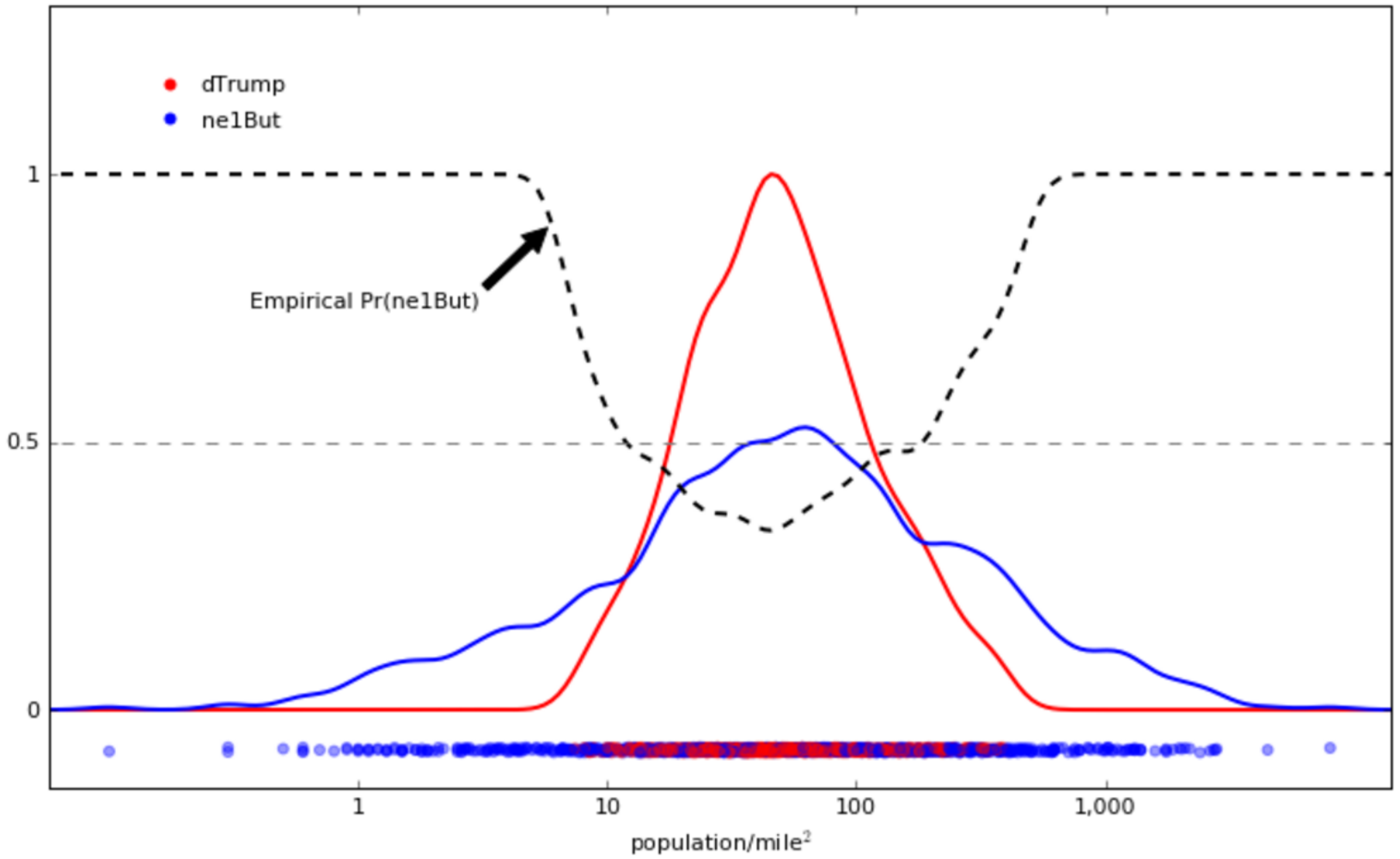
Slide Type Sub-Slide ▾

- Okay, let's look at some new data
- This is just a one dimensional plot in $\mathbb{R}^1$ (even though it looks like $\mathbb{R}^2$)
- We see that "average" population density counties favor Trump while "unusual" counties favor his opponents



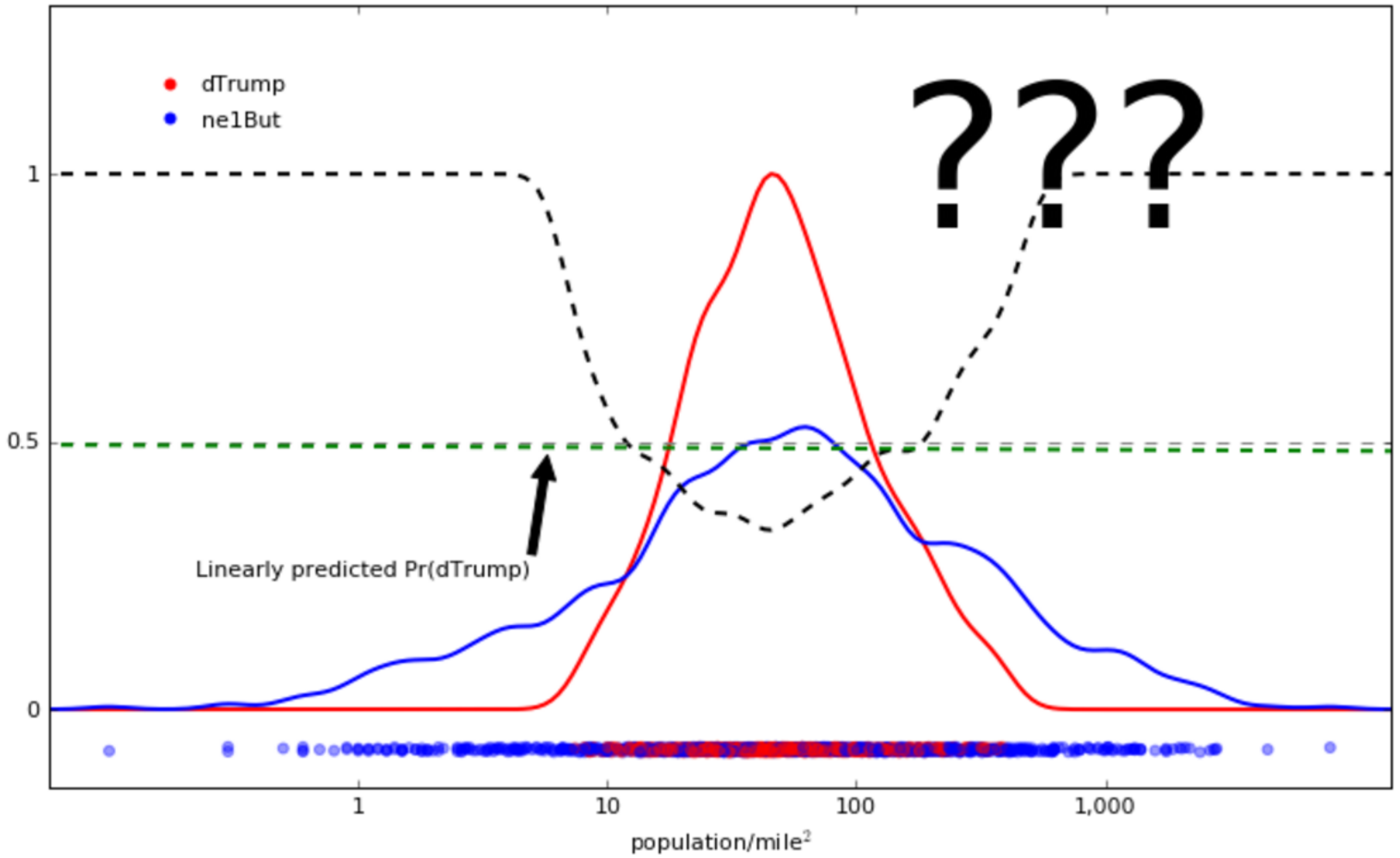
Slide Type Sub-Slide ▼

- Here are the relative "densities" of Trump and non-Trump counties
- It is as expected: "average" population density counties favor Trump while more extreme counties favor his opponents
- We should be able to leverage this information!



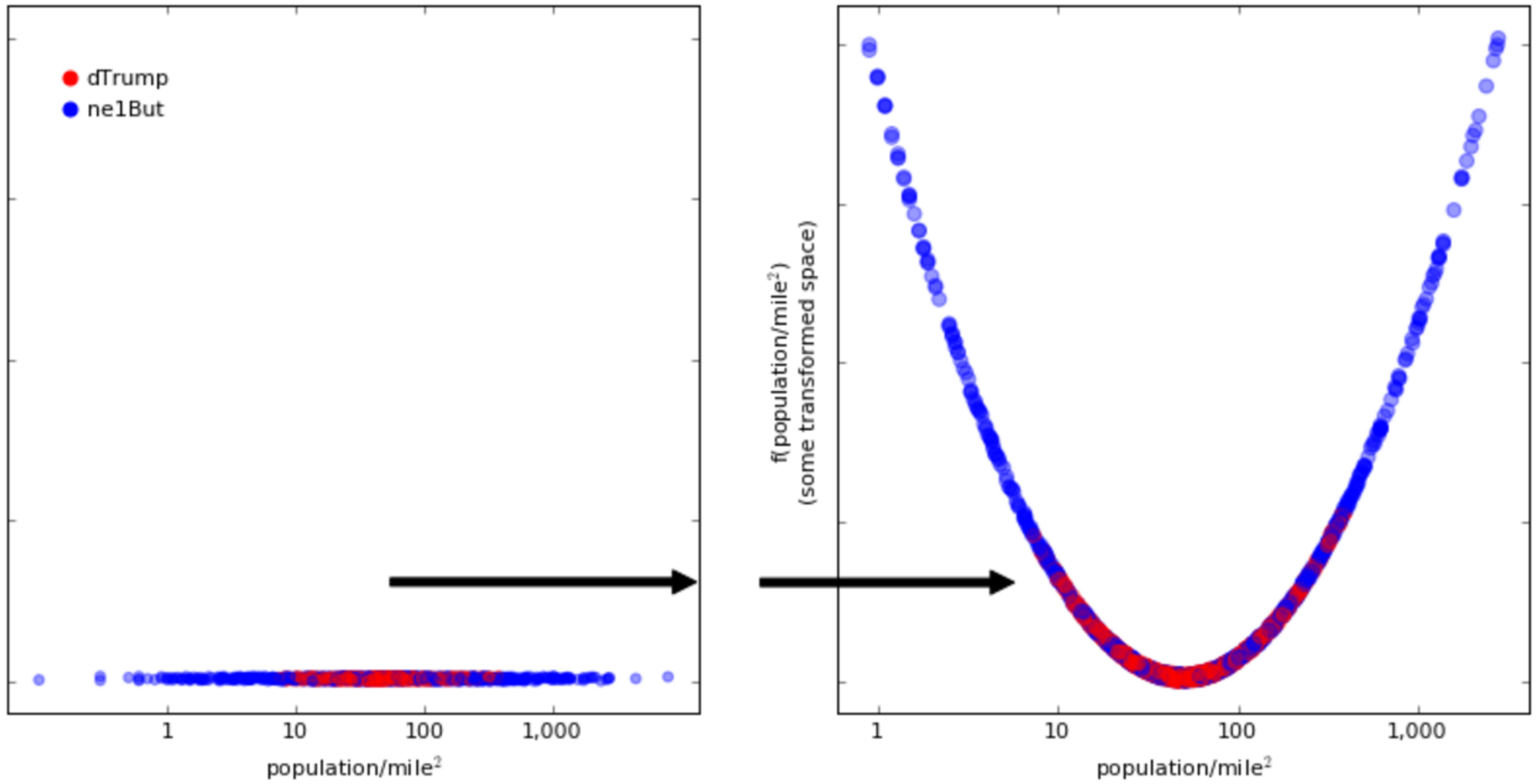
Slide Type Sub-Slide ▼

- Here we show the relative proportion non Trump counties to Trump counties across population density
- The 50/50 line is crossed when the relative propotion of Trump to non-Trump counties are equal
- Clearly there is predictive power in the population density variable



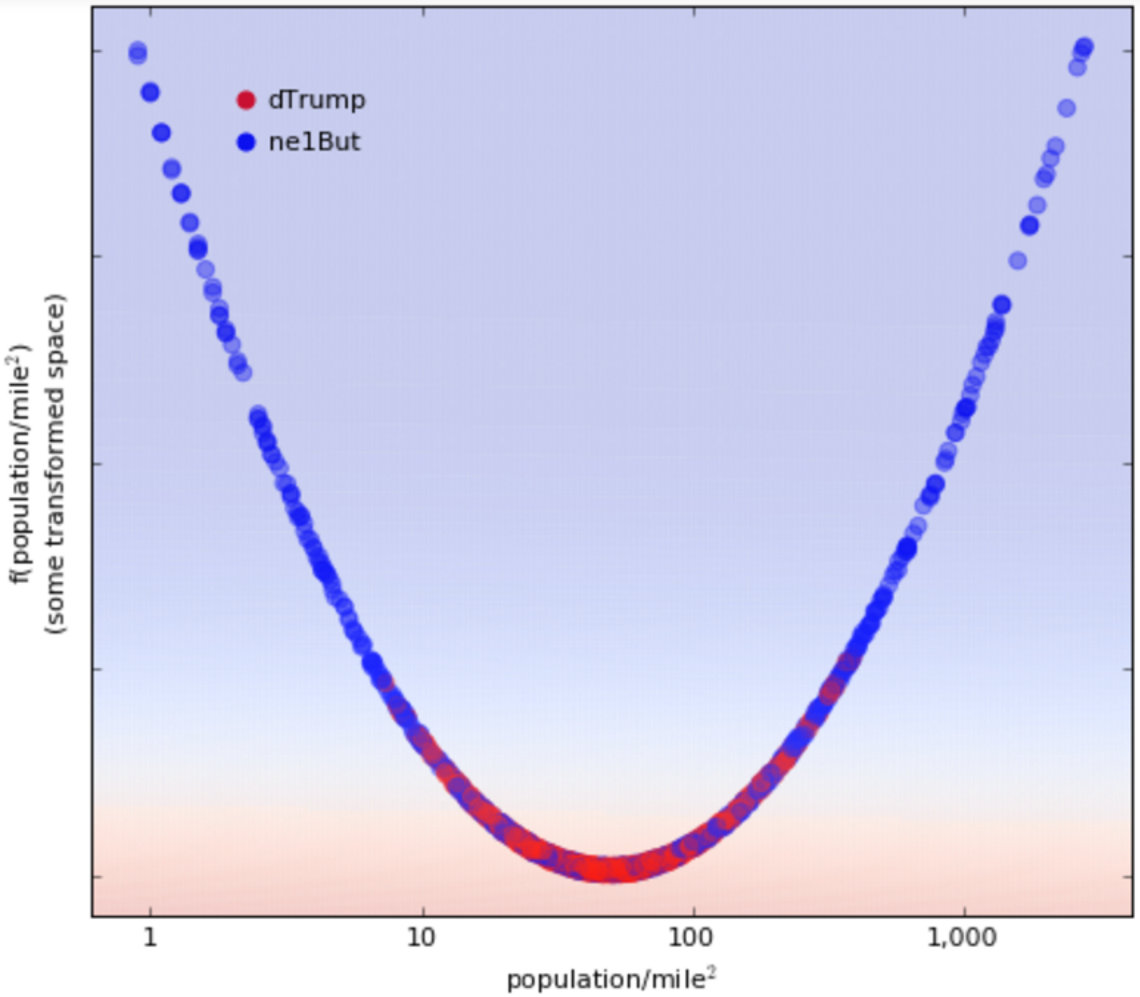
Slide Type Sub-Slide ▼

- However, when we fit our trusty old logistic regression on the population density variable, we find no predictive power
- This is becaues the relationship is not linear -- preference doesn't constantly [in/de]crease with population density
- The relationship is said to be *"non-linear"* in the original predictor space



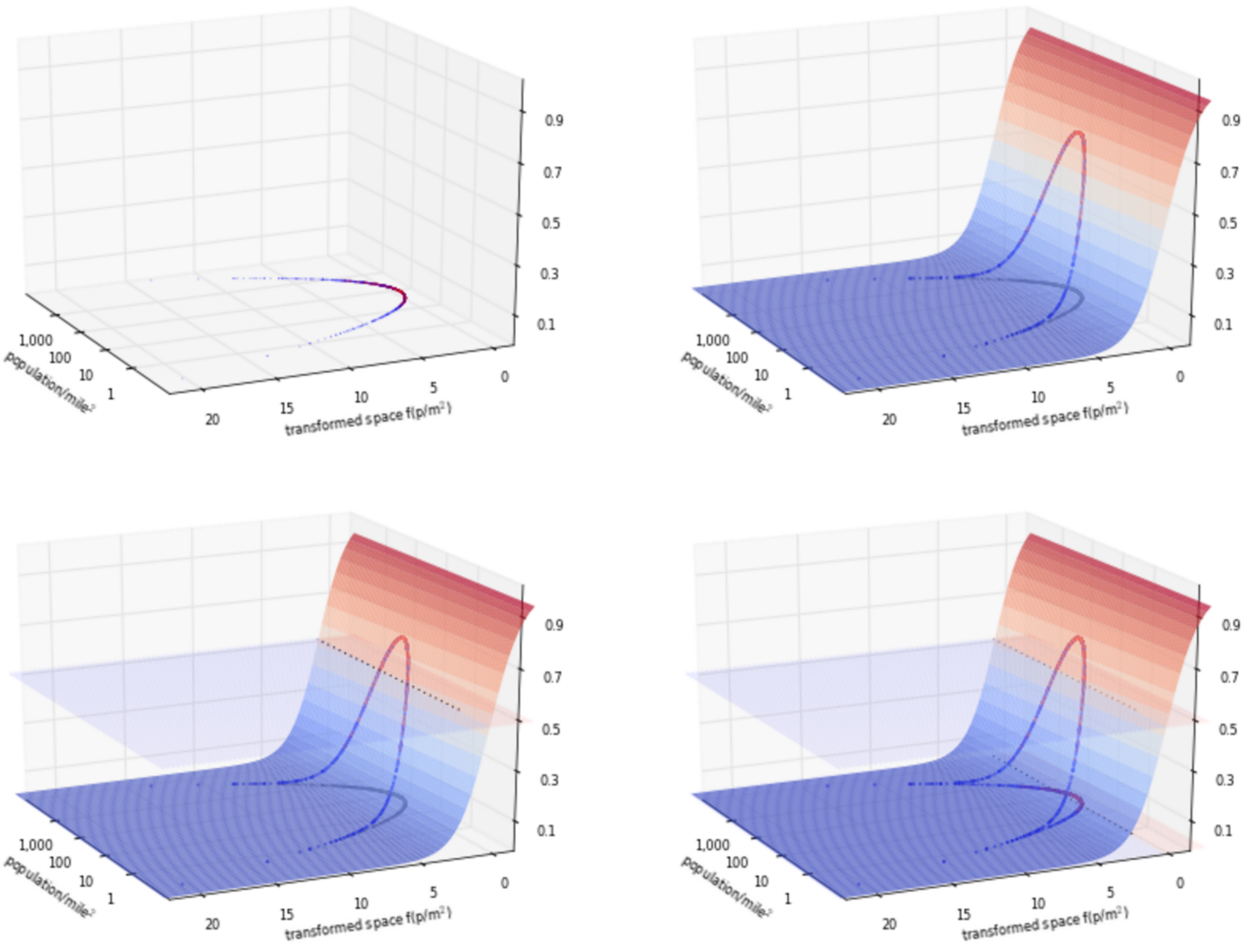
Slide Type Sub-Slide ▼

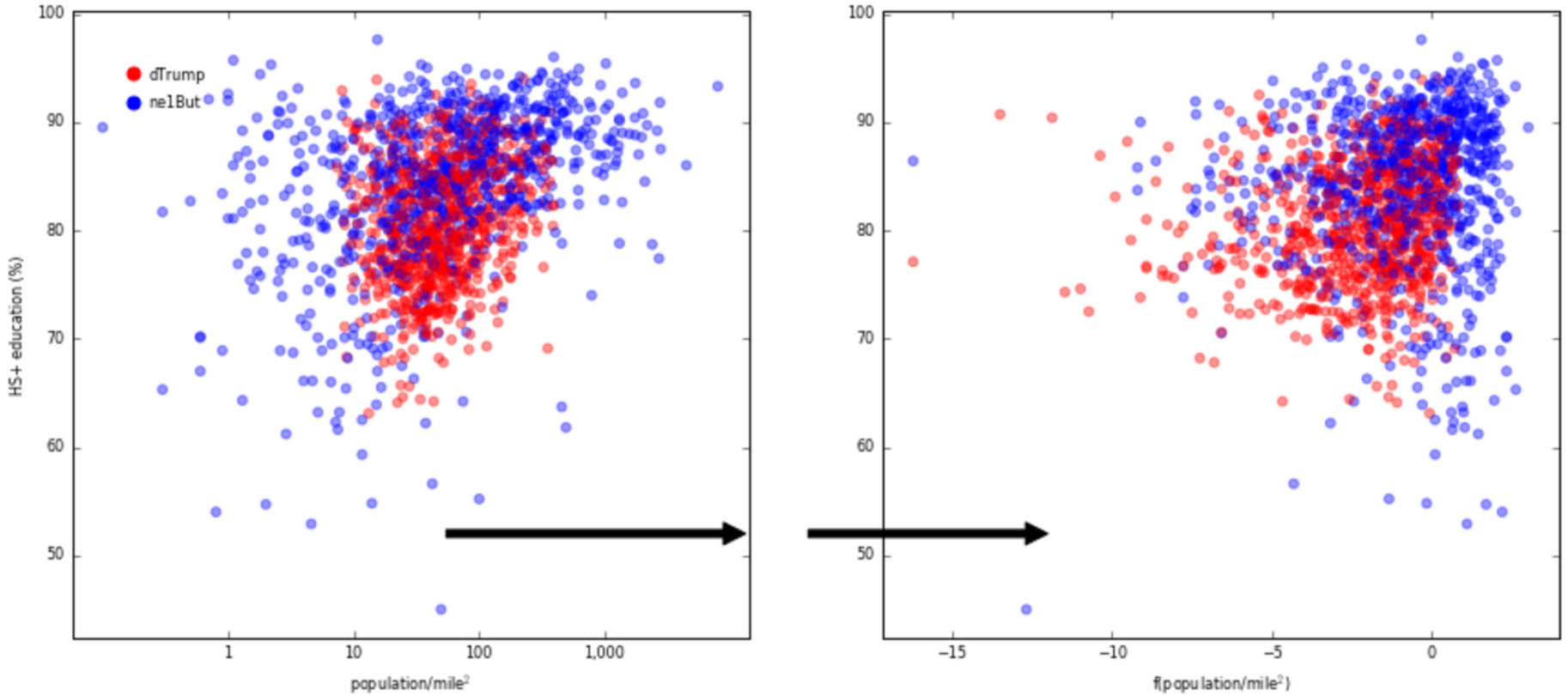
- So here's a super idea: let's convert (transform) the predictor into a new space were we can find a linear relationship
- What we're doing is embedding the original axis as a subspace in a new space -- here a "U" in a bivariate space
- But notice it's really still the same space, just bent/stretched/twisted into a new shape where linear machinary works



Slide Type Sub-Slide ▼

- Now we can see that we have a linear gradient (in the "y" direction)
- Logistic regression can now easily leverage the predictive power of this "linear" effect
- So it was a question of projecting the population density variable into a space where it's information was accessible


0.9
0.7
0.5
0.3
0.1
1,000
100
10
1
population/mile²
20
15
10
5
0
transformed space f(p/m²)



Slide Type Sub-Slide ▼

- What we have done is transform

$$\mathbf{x} = (x_1, x_2) \rightarrow \mathbf{z} = (f(x_1), x_2)$$

- Or previously

$$\mathbf{x} = (x_1) \rightarrow \mathbf{z} = (x_1, f(x_1))$$

- Indeed, for an arbitrary transform $f : \mathbb{R}^P \mapsto \mathbb{R}^Q$ we can map

$$\mathbf{x} \in \mathbb{R}^P \rightarrow \mathbf{z} \in \mathbb{R}^Q$$

- Regardless of if we use $\mathbf{x}$ or $\mathbf{z}$, the support vector classifier is estimated same way, namely

$$\mathbf{1}'\boldsymbol{\lambda} - \tfrac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda}, \quad Q_{mn} = y_m\mathbf{x}_m'\mathbf{x}_n y_n, \text{ or}$$
$$\mathbf{1}'\boldsymbol{\lambda} - \tfrac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda}, \quad Q_{mn} = y_m\mathbf{z}_m'\mathbf{z}_n y_n \quad \text{with} \begin{bmatrix} -I \\ -\mathbf{y} \\ \mathbf{y} \\ \mathbf{I} \end{bmatrix} \boldsymbol{\lambda} \leq \begin{bmatrix} \mathbf{0} \\ 0 \\ 0 \\ \mathbf{C} \end{bmatrix}$$

- Notice that this only depends on the inner product $\mathbf{x}_1'\mathbf{x}_2$ or $\mathbf{z}_1'\mathbf{z}_2$
- Now suppose I could produce a function K of just $\mathbf{x}_1, \mathbf{x}_2$ that was equal to the inner product $\mathbf{z}_1'\mathbf{z}_2$

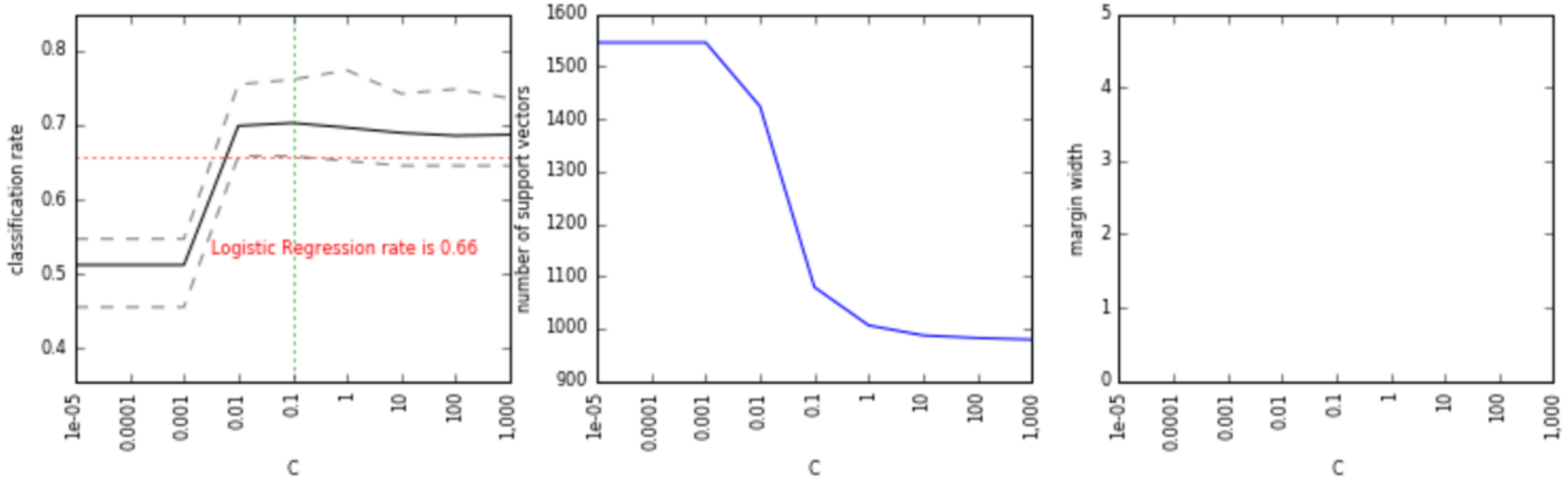
$$\mathbf{z}_1'\mathbf{z}_2 = f(\mathbf{x}_1)'f(\mathbf{x}_2) = K(\mathbf{x}_1, \mathbf{x}_2)$$

- for some map $f: \mathbf{x} \in \mathbb{R}^P \mapsto \mathbf{z} \in \mathbb{R}^Q$
- Such a function is called a *kernel*
- Kernels allows us to compute *inner products* in $f(\mathbf{x})$ space without actually having to visiting the space
- This is called the *"kernel trick"*
- *Are kernels even real??* (Yes, here are a couple examples)

$$f(x) = (x_1, x_1^2)$$

$$f(x_1)'f(x_2) = x_1x_2 + x_1^2x_2^2 = K(x_1, x_2)$$

$$\begin{aligned} f(x_1)'f(x_2) &= \exp\left(-|x_1 - x_2|^2\right) \\ &= \exp\left(-x_1^2 + 2x_1x_2 - x_2^2\right) \\ &= \exp\left(-x_1^2\right)\exp\left(-x_2^2\right)\exp(2x_1x_2) \\ &= a_1a_2\sum_{k=0}^{\infty}\frac{(2x_1x_2)^k}{k!} \\ &= a_1\left[\sqrt{\tfrac{2^0}{0!}}x_1^0, \quad \sqrt{\tfrac{2^1}{1!}}x_1^1, \quad \cdots\right] a_2 \begin{bmatrix} \sqrt{\frac{2^0}{0!}}x_2^0 \\ \sqrt{\frac{2^1}{1!}}x_2^1 \\ \vdots \end{bmatrix} \end{aligned}$$



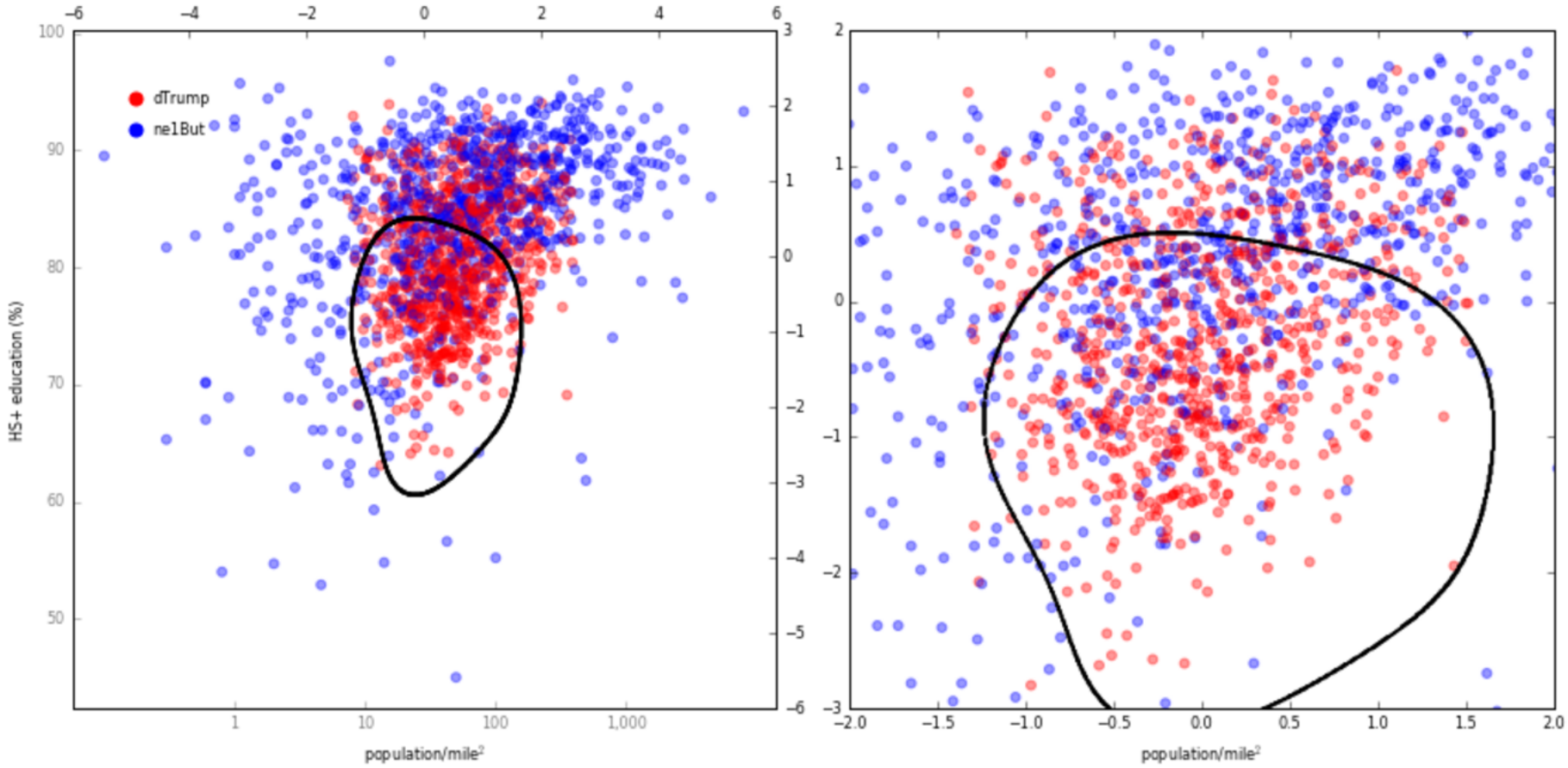
Slide Type Sub-Slide ▾

- We specified the *"radial basis function"* kernel (the second example kernel from above)
- This projects our data into a space with an infinite number of dimensions (!?!?)
  - (This new space is big! And a highly non-linear transformation of the original space)
  - (remember our "U"-bending transformation?)

- Because this space is so large we actually probably **could** find a hyperplane that linearly separates the data!!
  - so all the "soft margin" stuff *(Where is it, anyway?)* -- saying "oh, data isn't linearly separable" -- was moot!
  - ...except of course it does provide regularization... so there's that...

- When we project into this new space we beat the logistic regression
  - (and all our hard work to transform the variables to allow us to exploit linear relationships)
  - And we didn't even have to think about the transformation that was happening!!

- We seemingly confronted the "curse of dimensionality" head on, **and won** -- inducing no overfitting!



Slide Type Sub-Slide

Neat things we covered

- **SVCs** provide a "best" separating hyperplane that maximizes the "margin" between classes
  - they are an elegant solution to the classification problem
- **Soft Margin SVCs** use a penalty C to control the amount of violation of the margin

Slide Type Sub-Slide ▼

Neat things we covered

- **SVCs** provide a "best" separating hyperplane that maximizes the "margin" between classes
  - they are an elegant solution to the classification problem
- **Soft Margin SVCs** use a penalty C to control the amount of violation of the margin
  - Increasing the margin regularizes estimation via the separating planes reliance on blunt data characteristics
- **Cross-Validation** can be used to tune C and assess model estimate variation
  - This allows us to assess overfitting
- **the *"Kernel Trick"*** projects our prediction space into another space where we can calculate the inner product *using only the variables of the original space*
  - this allows us to readily leverage non-linearly transformed variable spaces
  - and makes SVMs powerful and flexible class of "out of the box" solutions

Slide Type Sub-Slide ▼

Neat things we didn't cover

- **loss functions and SGD**
  - SVRs, etc.
- **Multiclass Classifiers**
  - one-versus-many and one-versus-one
- **Kernels**
  - you know you wanna...

```
from sklearn.linear_model import SGDClassifier
```

Slide Type Sub-Slide