# APMA E4990.02: Classification Problems

Lecture 5

# Examples of Classification Problems

- User churn for subscription services.
- Acquisition - who is likely to subscribe?
- Who should receive a loan?
- Engagement - Who is likely to click? How do we target them? (later)
- Purchases - will somebody purchase a product? How will they rate it?




# Outline

- Logistic regression hypothesis
- Decision Boundary
- Cost function (why we need a new one)
- Gradient Descent and a concrete example in Python
- AUC/ROC and model evaluation.
- Model selection and regularization for classification problems.

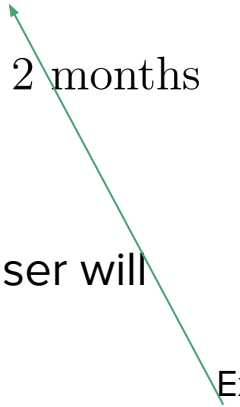
# Example of a Classification Problem - User Churn

The New York Times

$X_n = [\text{'subscription tenure', 'age', 'online activity', 'authors', etc}]$

$y = \text{probability of ending subscription in the next 2 months}$

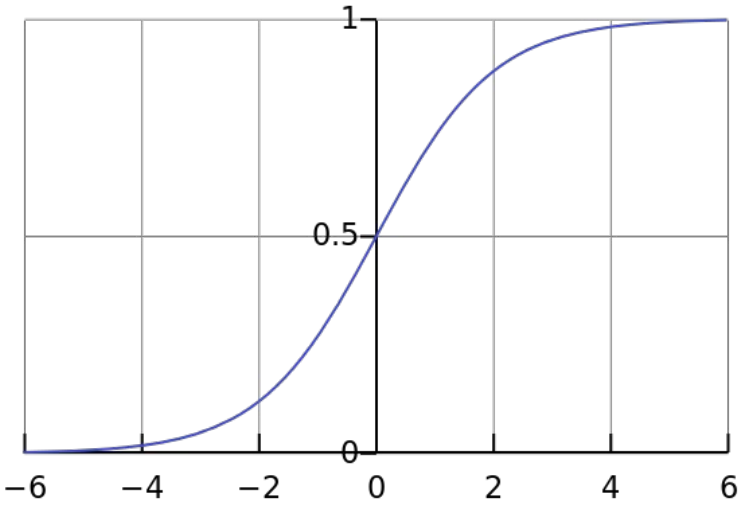
**Question:** How do we predict the probability a user will churn?



Example of some of the most predictive features for the churn model at The New York Times

# What is the model?

$$\log \frac{p(y=1)}{1-p(y=1)} = \beta \cdot \mathbf{x}$$

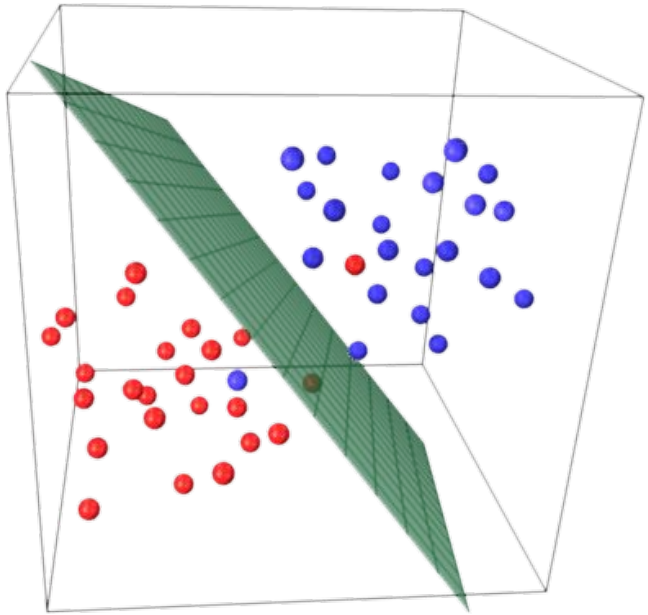



- p is the probability of the outcome.
- We compose the output of a linear model with a function which has range between -1 and 1.
- Beta defines a decision boundary in the variable space (y=1, y=0).

$$p_\beta(\mathbf{x}) = \frac{1}{1+\exp(-\beta^T \cdot \mathbf{x})}$$

# Decision boundary for Logistic Regression

$$p_\beta(\mathbf{x}) = \frac{1}{1 + \exp(-\beta^T \cdot \mathbf{x})}$$



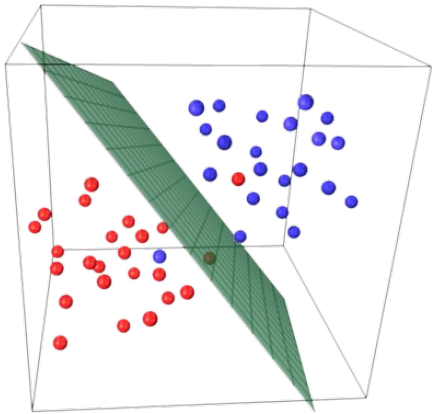
- This defines a decision boundary where:

$\beta^T \cdot \mathbf{x}$

- Large and positive implies class 1
- Large and negative implies class 0

# Decision boundary for Logistic Regression

$$p_\beta(\mathbf{x}) = \frac{1}{1 + \exp(-\beta^T \cdot \mathbf{x})}$$



Let R denote the red class and B the blue class. Then

$$\text{If } p_\beta(\mathbf{x}) > 0.5 \text{ then } \mathbf{x} \in R$$

$$\text{If } p_\beta(\mathbf{x}) \leq 0.5 \text{ then } \mathbf{x} \in B$$

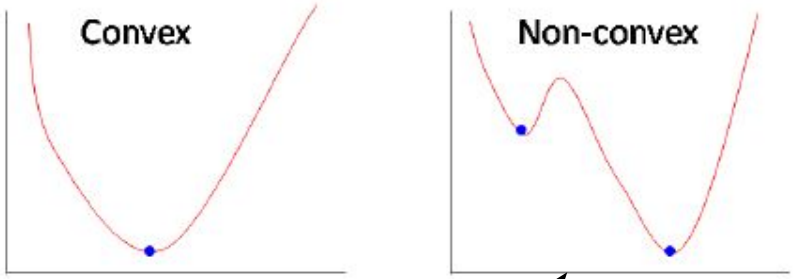
**Note:** The choice of 0.5 here is arbitrary.

# The Logistic Regression Cost Function

# How do we measure distance?

For linear regression, recall we used L^p norms. However for the logit function, when we need a different way to measure performance of our model.

$$p_\beta(\mathbf{x}) = \frac{1}{1 + \exp(-\beta^T \cdot \mathbf{x})}$$

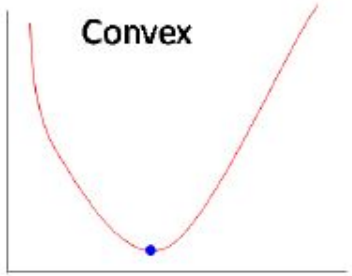


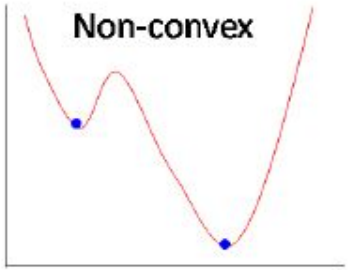

$$\text{Cost}_1(h_\beta(\mathbf{x}), \mathbf{y}) := \frac{1}{N} \sum_{k=1}^{N} (p_\beta(x) - y)^2$$

We would obtain a **non-convex cost function.** These are **difficult to minimize.** Instead we define another cost function (its derivation will be explained in more detail when we get into probabilistic concepts)

# How do we measure distance? The new cost

$$p_\beta(\mathbf{x}) = \frac{1}{1 + \exp(-\beta^T \cdot \mathbf{x})}$$




$$\text{Cost}_2(p_\beta(\mathbf{x}), y) = \begin{cases} -\log p_\beta(\mathbf{x}) & \text{if } y \text{ is } 1 \\ -\log(1 - p_\beta(\mathbf{x})) & \text{if } y \text{ is } 0 \end{cases}$$

- When y is 1, $p_\beta(\mathbf{x})$ tries to be close to 1.
- When y is 0, $p_\beta(\mathbf{x})$ tries to be close to 0.
- It may seem like we pulled this out of thin air, but we didn't. It actually can be **derived naturally from the method of Maximum Likelihood**.

# The probabilistic approach - maximum likelihood.

$$p = \frac{\exp(\beta \cdot \mathbf{x})}{1 + \exp(\beta \cdot \mathbf{x})}$$

We then compute the likelihood function, which is just the conditional probability of the outcome on the data.

$$\mathcal{L}(\beta|x_n, y_n) = \prod_{k=1}^{N} p^{y_k}(1-p)^{1-y_k}$$

$$l(\beta|x, y) = \sum_{k=1}^{n} y_k \log p(x_k) + (1-y_k)\log(1-p(x_k))$$

# Relating likelihood to our cost

$$\text{Cost}_2(p_\beta(\mathbf{x}), y) = \begin{cases} -\log p_\beta(\mathbf{x}) & \text{if } y \text{ is } 1 \\ -\log(1 - p_\beta(\mathbf{x})) & \text{if } y \text{ is } 0 \end{cases}$$

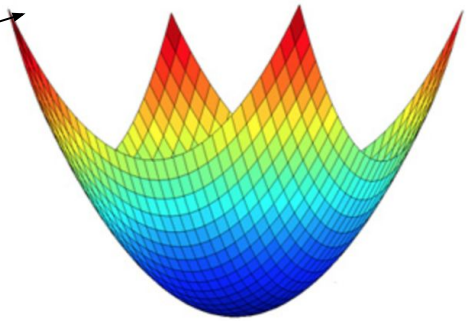
$$l(\beta|\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{n} y_k \log p_\beta(\mathbf{x_k}) + (1 - y_k) \log(1 - p_\beta(\mathbf{x_k}))$$

# How do we maximize this?

$$\prod_{k=1}^{N} p^{y_k}(1-p)^{1-y_k} = \prod_{k=1}^{N} \frac{\exp(y_k \beta \cdot \mathbf{x})}{1+\exp(\beta \cdot \mathbf{x})}$$

$$l(\beta|x,y) = \sum_{k=1}^{n} y_k \log p(x_k) + (1-y_k)\log(1-p(x_k))$$

Claim: $\beta \mapsto l(\beta|x,y)$ is convex



# Setting up Gradient Descent

$$l(\beta|x,y) = \sum_{k=1}^{n} y_k \log p(x_k) + (1-y_k)\log(1-p(x_k))$$

$$\frac{\partial l}{\partial \beta_i} = \sum_{k=1}^{N} (y_k - p(x_k))x_{ki}$$

Notice that when **y is 1, then p should be 1**, and when **y is 0, p should be 0.**

$$\frac{\partial^2 l}{\partial \beta_i \partial \beta_j} = -\sum_{k=1}^{n} p(x_k)(1-p(x_k))x_{ki}x_{kj}$$

# Why we need this?

- This is our first encounter with a **minimization problem that doesn't have an exact solution** - hence we **need to use gradient descent to reach a minimum.**

- The **method of maximum likelihood is used to solve almost all parametric problems in machine learning**. We will cover this more thoroughly in a few lectures.

# Concrete Example

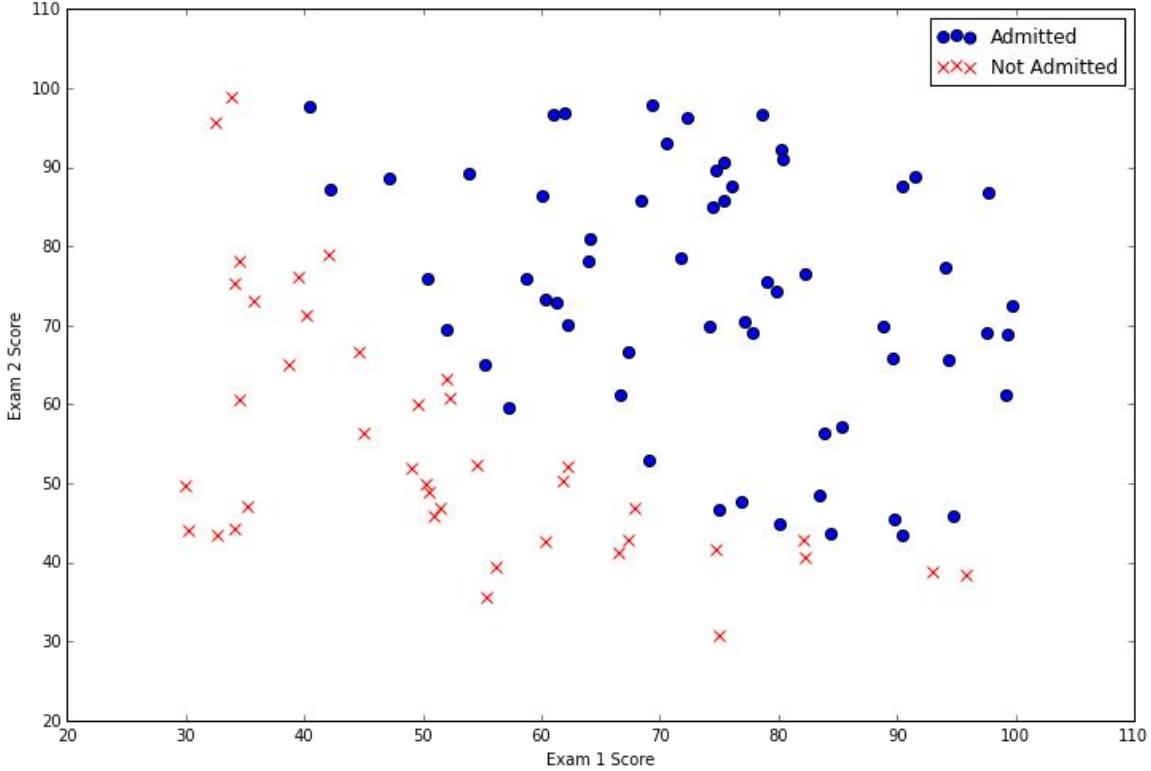
Admission based on test scores

# Example - Admission to a program

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import os
path = os.getcwd() + '\data\ex2data1.txt'
data = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2', 'Admitted'])
data.head()
```

| | Exam 1 | Exam 2 | Admitted |
|---|---|---|---|
| 0 | 34.623660 | 78.024693 | 0 |
| 1 | 30.286711 | 43.894998 | 0 |
| 2 | 35.847409 | 72.902198 | 0 |
| 3 | 60.182599 | 86.308552 | 1 |
| 4 | 79.032736 | 75.344376 | 1 |

# Scatter plot of admission results



# Python code

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

```
def cost(theta, X, y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    first = np.multiply(-y, np.log(sigmoid(X * theta.T)))
    second = np.multiply((1 - y), np.log(1 - sigmoid(X * theta.T)))
    return np.sum(first - second) / (len(X))
```

$$\frac{\partial l}{\partial \beta_i} = \sum_{k=1}^{N} (y_k - p(x_k)) x_{ki}$$

```
def gradient(theta, X, y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)

    parameters = int(theta.ravel().shape[1])
    grad = np.zeros(parameters)

    error = sigmoid(X * theta.T) - y

    for i in range(parameters):
        term = np.multiply(error, X[:,i])
        grad[i] = np.sum(term) / len(X)

    return grad
```

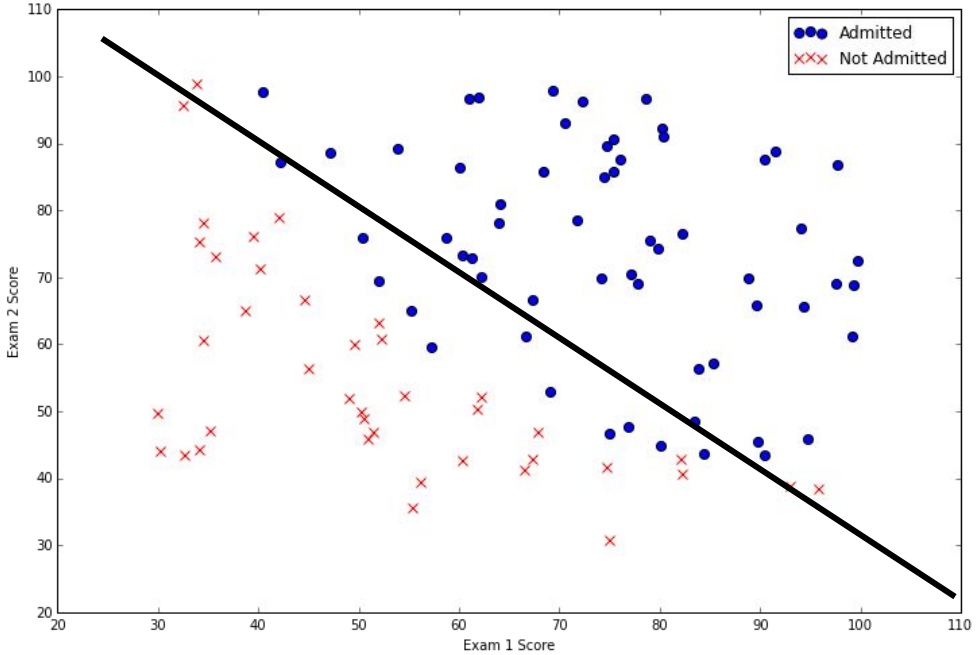
# Minimizing Cost in Python

$$\text{Cost}_2(p_\beta(\mathbf{x}), y) = \begin{cases} -\log p_\beta(\mathbf{x}) & \text{if } y \text{ is } 1 \\ -\log(1 - p_\beta(\mathbf{x})) & \text{if } y \text{ is } 0 \end{cases}$$

```
import scipy.optimize as opt
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(X, y))
cost(result[0], X, y)
```

*0.20357134412164668*

# Final Decision Boundary




$$p_\beta(\mathbf{x}) = \frac{1}{1 + \exp(-\beta^T \cdot \mathbf{x})}$$

# How do we measure performance of a classifier?

# Accuracy

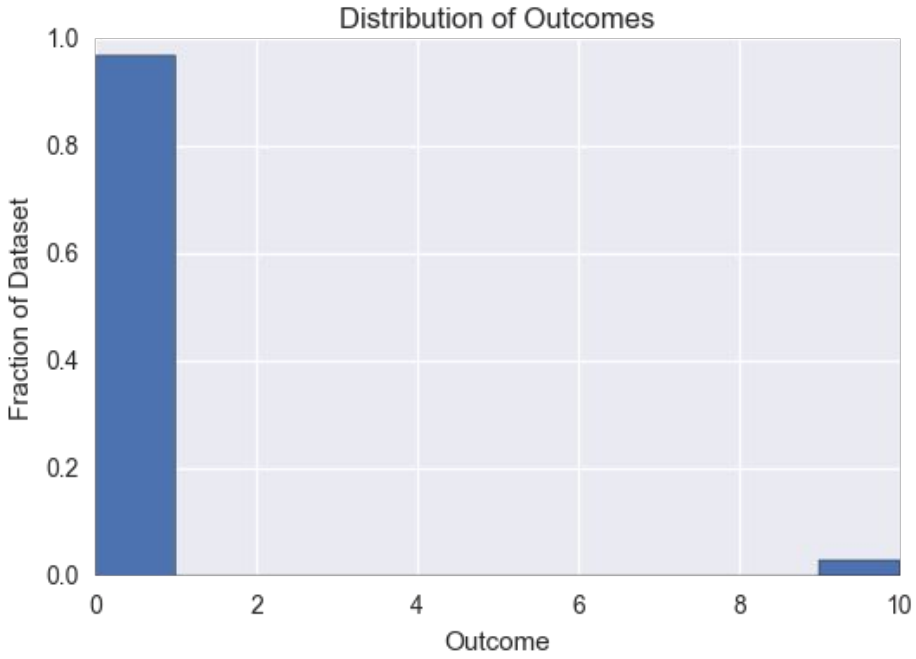
$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(y_i = f(x_i))$$

- # of correct predictions / # of samples.

Easy to understand, but not a good metric in general. Why?

# What would be a good model here?

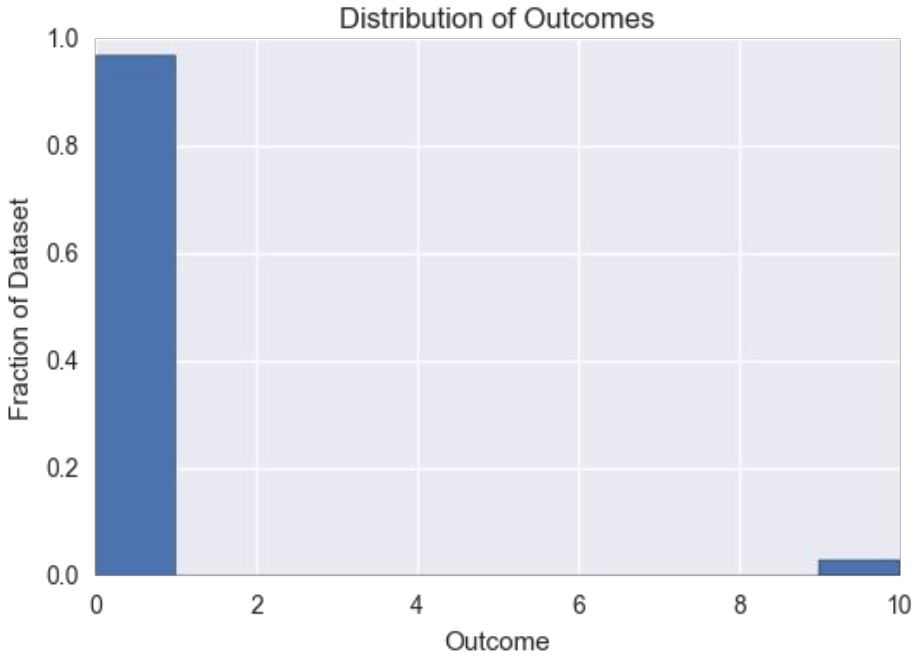
$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(y_i = f(x_i))$$



# What would be a good model here?

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(y_i = f(x_i))$$

- If we just set **y = 0**, we will have **97% accuracy!**
- Accuracy is a useless measure unless you break it into deciles or have a balanced dataset.



# What’s wrong with accuracy?

- In any scenario I’ve encountered thus far, datasets are *heavily* imbalanced.This means one outcome occurs much more frequently than the alternative. **Examples are:** users subscribing, churning, clicking etc.

- In general, you have **thousands, if not millions of impressions**, but maybe only **1-10% (or less) of the outcome variable** you are interested in.

- If we create a model which simply sets y=0, we will have incredible accuracy - what does that mean though?

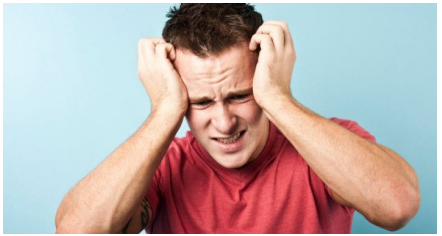
# Primary Methods Used

For measuring the performance of models,

There are two standards:

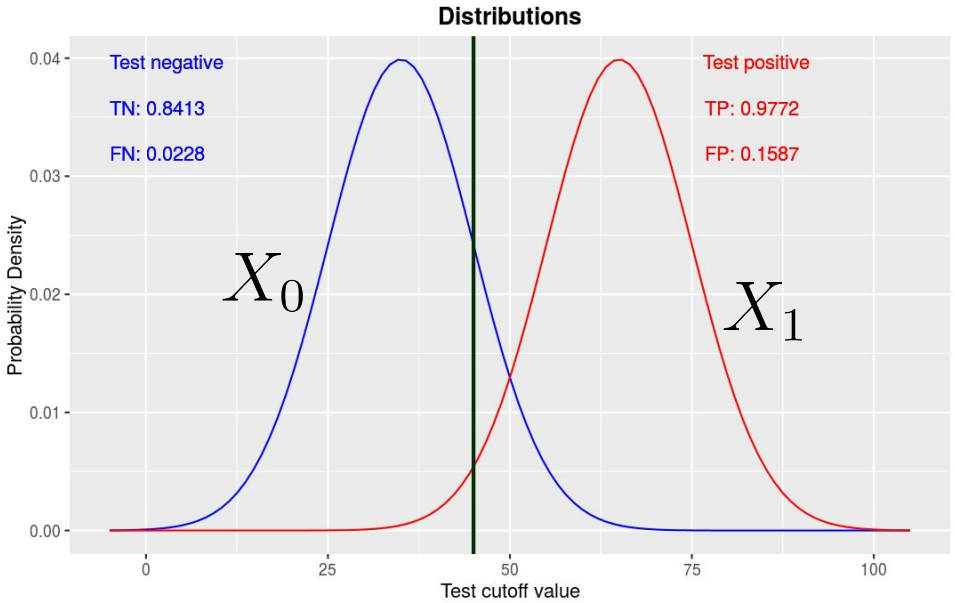
- AUC ROC (area under ROC curve) .
  (good for scientists)

- Accuracy as a function of percentile of propensity.
  (good for stakeholders)



# AUC ROC and Lift Curves

# Plotting a classifier

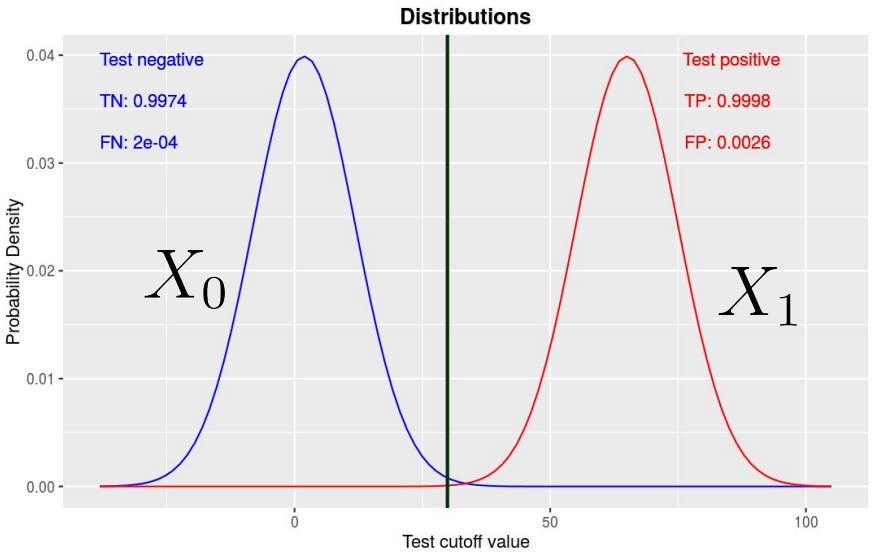



$$X_1 = p_\beta(x|y=1)$$

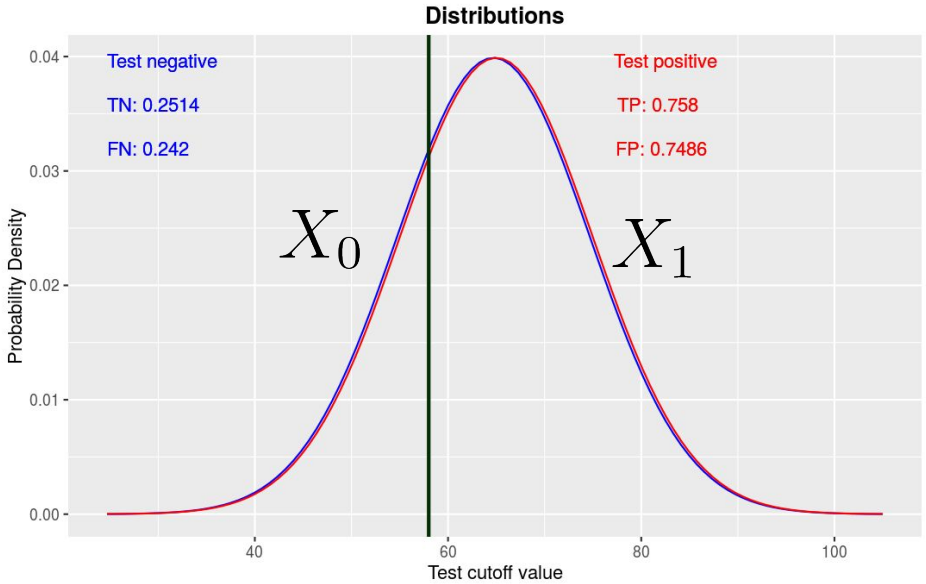
$$X_0 = p_\beta(x|y=0)$$

- Intuitively speaking, we wish to somehow measure **how well our classifier can separate these two distributions.**

# Perfect versus Random

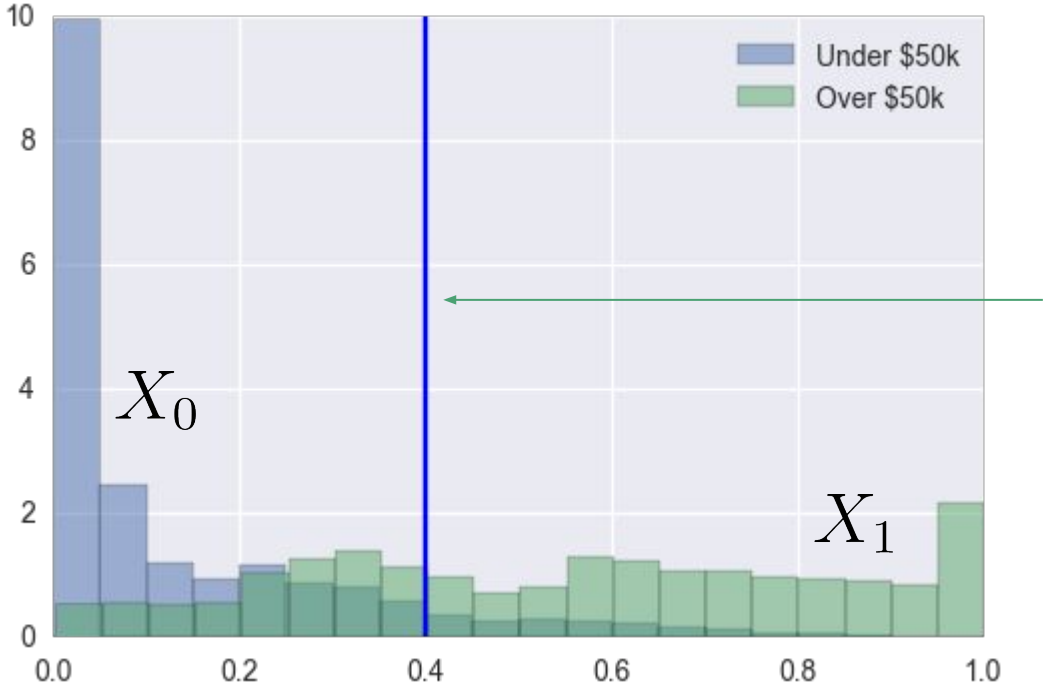



A perfect classifier is able to separate the two classes (1 and 0) completely.




A completely random classifier can't make any distinction.

# Adult Data Set Example




Probability Output from Our Model

- A good classifier is one which separates the two distributions well.
- After assigning ‘scores’, we often want to find a good threshold to separate the classes. How do we do this?

# Definitions

- We **predict 0** while we should have the **class is actually 0**: this is called a **True Negative**, i.e. we correctly predict that the class is negative (0). For example, an antivirus did not detect a harmless file as a virus .

- We **predict 0** while we should have the **class is actually 1**: this is called a **False Negative**, i.e. we incorrectly predict that the class is negative (0). For example, an antivirus failed to detect a virus.

- *We **predict 1** while we should have the **class is actually 0**: this is called a **False Positive**, i.e. we incorrectly predict that the class is positive (1). For example, an antivirus considered a harmless file to be a virus.*

- *We **predict 1** while we should have the **class is actually 1**: this is called a **True Positive**, i.e. we correctly predict that the class is positive (1). For example, an antivirus rightfully detected a virus.*

# Confusion Matrix

| | | Predicted class | |
|---|---|---|---|
| | | Class 1 | Class 0 |
| Actual class | Class 1 | 10 true positives (TP) | 2 false negatives (FN) |
| | Class 0 | 3 false positives (FP) | 35 true negatives (TN) |

In this example of a confusion matrix, among the **50 data points** that are classified, **45 are correctly classified** and the **5 are misclassified.**

# True positive and False Positive Rates

- True positive rate (**TPR**), aka. sensitivity, hit rate, and recall, which is defined as

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

- Intuitively this metric corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. In other words, the higher TPR, the fewer positive data points we will miss.
- False positive rate (**FPR**), aka. fall-out, which is defined as

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

- Intuitively this metric corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points. In other words, the higher FPR, the more negative data points we will misclassified.

# An example calculation

| | | Predicted class | |
|---|---|---|---|
| | | Class 1 | Class 0 |
| Actual class | Class 1 | 10 true positives (TP) | 2 false negatives (FN) |
| | Class 0 | 3 false positives (FP) | 35 true negatives (TN) |

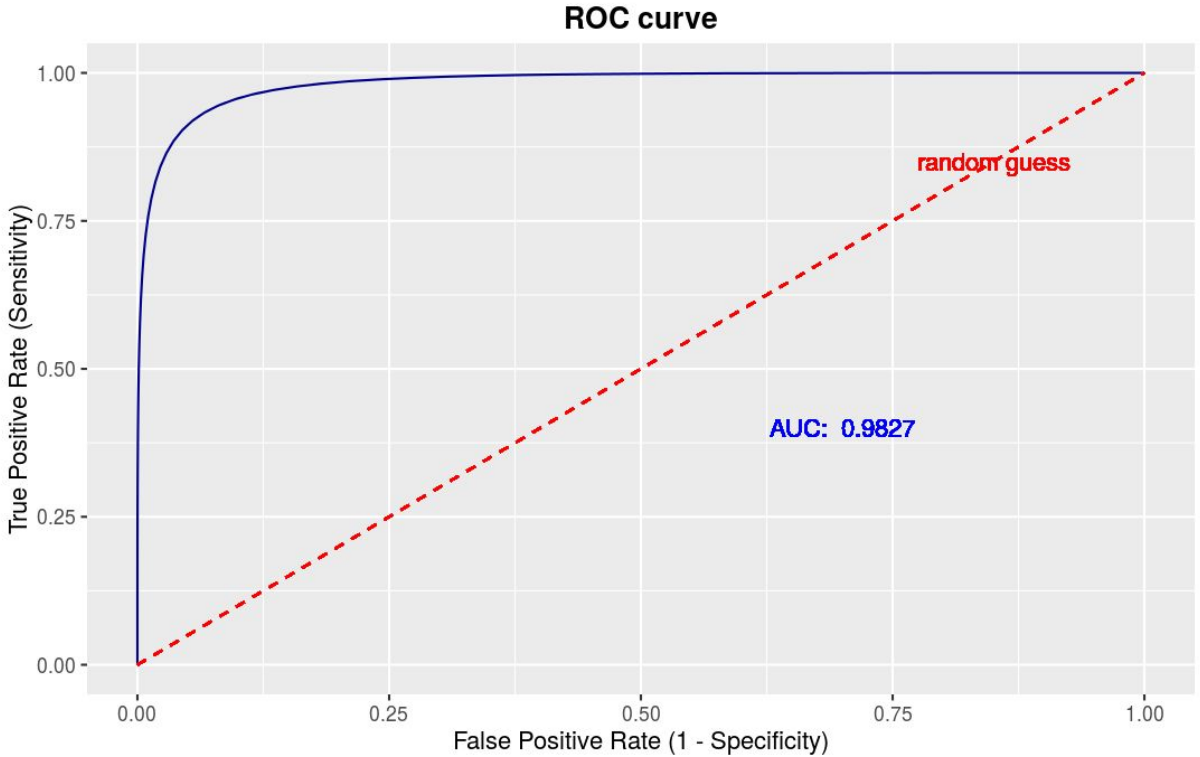
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = \frac{10}{10 + 2}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = \frac{3}{3 + 35}$$

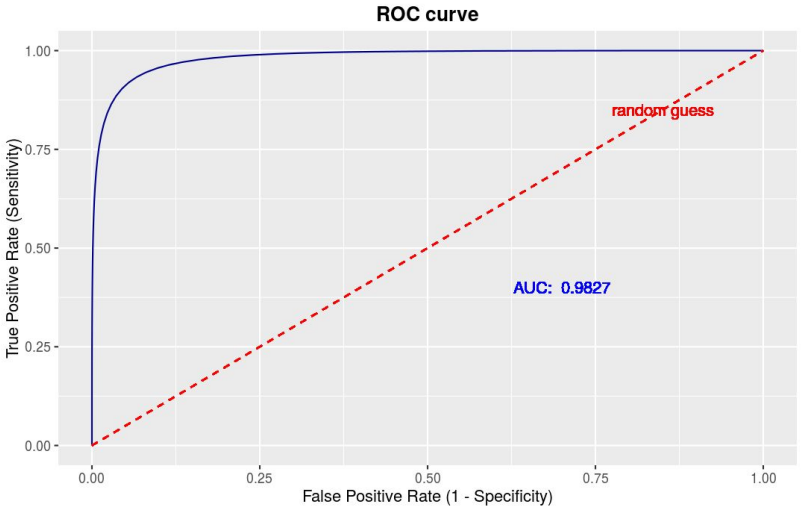
# Confusion Matrices

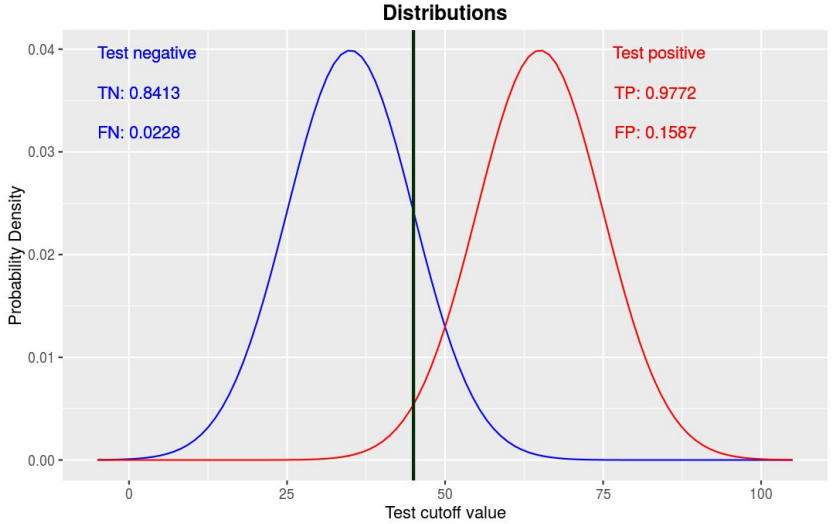
| A | B | C | C′ |
|---|---|---|---|
| TP=63 FN=37 100<br>FP=28 TN=72 100<br>91 109 200 | TP=77 FN=23 100<br>FP=77 TN=23 100<br>154 46 200 | TP=24 FN=76 100<br>FP=88 TN=12 100<br>112 88 200 | TP=76 FN=24 100<br>FP=12 TN=88 100<br>88 112 200 |
| TPR = 0.63 | TPR = 0.77 | TPR = 0.24 | TPR = 0.76 |
| FPR = 0.28 | FPR = 0.77 | FPR = 0.88 | FPR = 0.12 |
| PPV = 0.69 | PPV = 0.50 | PPV = 0.21 | PPV = 0.86 |
| F1 = 0.66 | F1 = 0.61 | F1 = 0.22 | F1 = 0.81 |
| ACC = 0.68 | ACC = 0.50 | ACC = 0.18 | ACC = 0.82 |

# ROC Curve

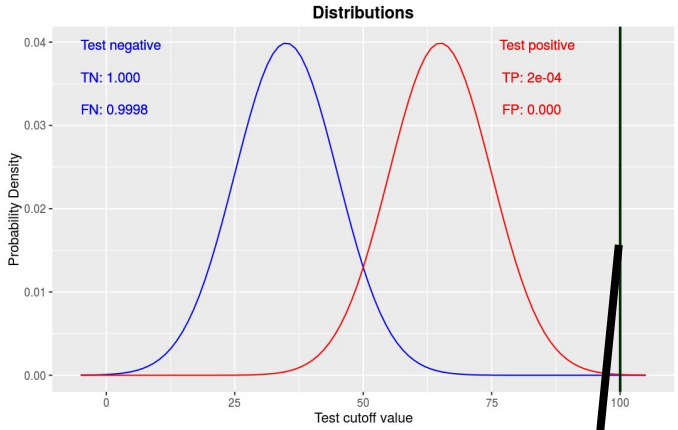

ROC curve
True Positive Rate (Sensitivity)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
False Positive Rate (1 - Specificity)
random guess
AUC: 0.9827

# ROC Curve

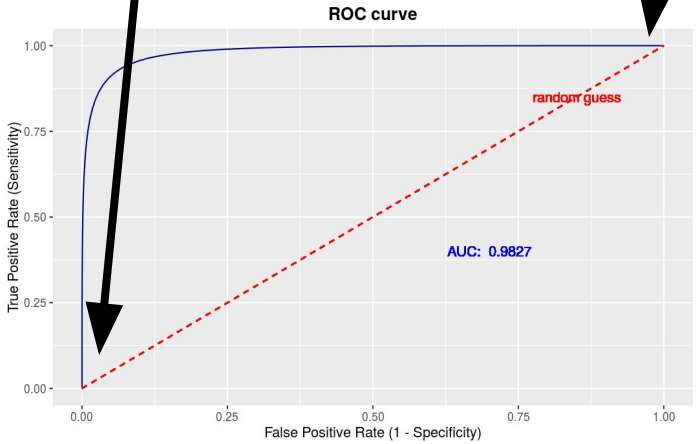

ROC curve
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
random guess
AUC: 0.9827
0.00
0.25
0.50
0.75
1.00

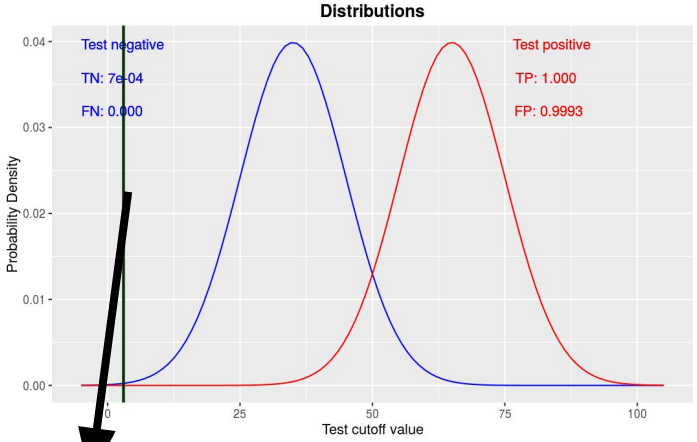

Distributions
Test negative
TN: 0.8413
FN: 0.0228
Test positive
TP: 0.9772
FP: 0.1587
Probability Density
Test cutoff value
0.00
0.01
0.02
0.03
0.04
0
25
50
75
100

# Extreme thresholds



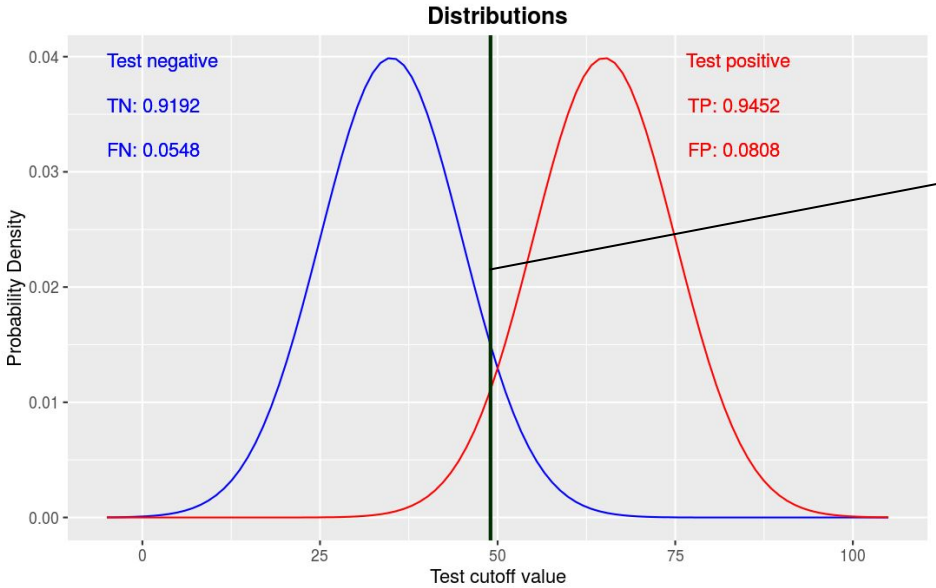

- Need 100% probability to be the red class.
- Assigns 0 to everything.

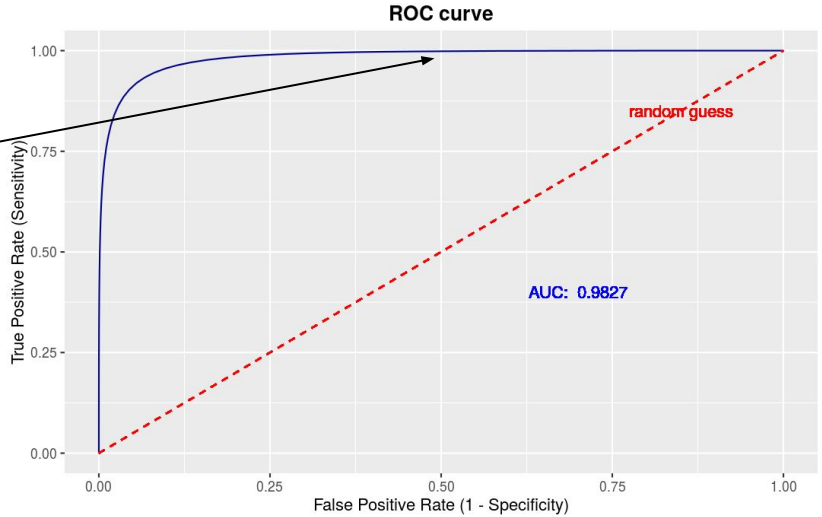


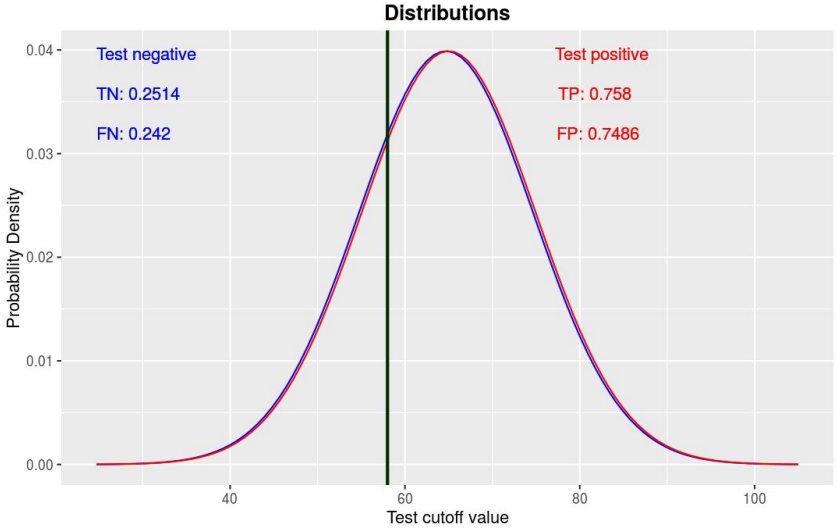

- Need above 0% probability to be in the red class.
- Assigns 1 to everything.

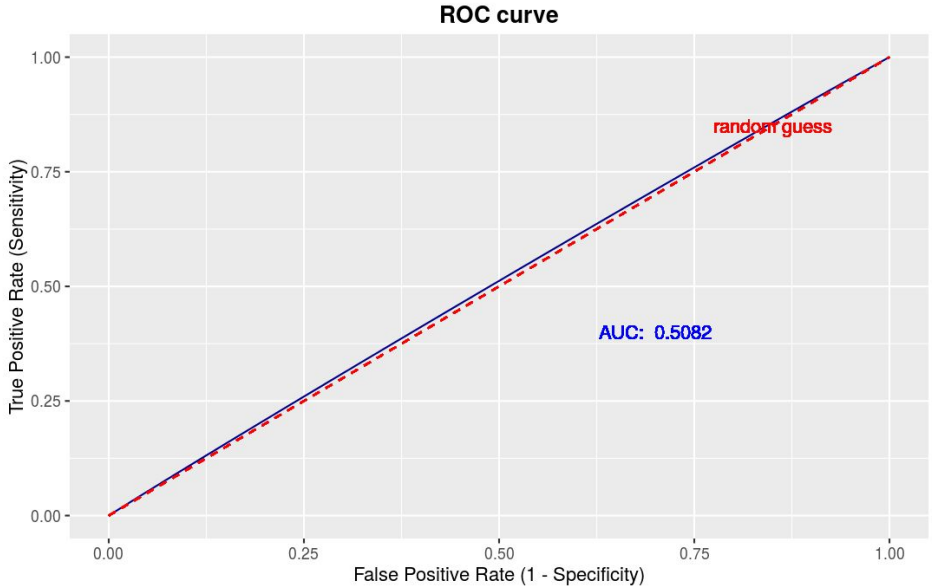
# The 50% mark


Distributions
Test negative
TN: 0.9192
FN: 0.0548
Test positive
TP: 0.9452
FP: 0.0808
Probability Density
0.04
0.03
0.02
0.01
0.00
0
25
50
75
100
Test cutoff value


ROC curve
True Positive Rate (Sensitivity)
1.00
0.75
0.50
0.25
0.00
random guess
AUC: 0.9827
0.00
0.25
0.50
0.75
1.00
False Positive Rate (1 - Specificity)

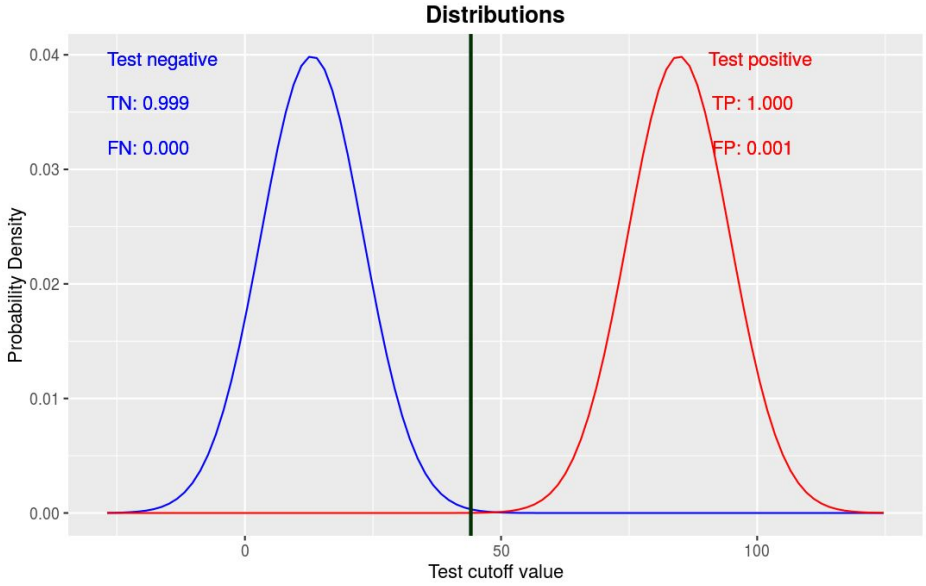
# ROC of a Random Classifier

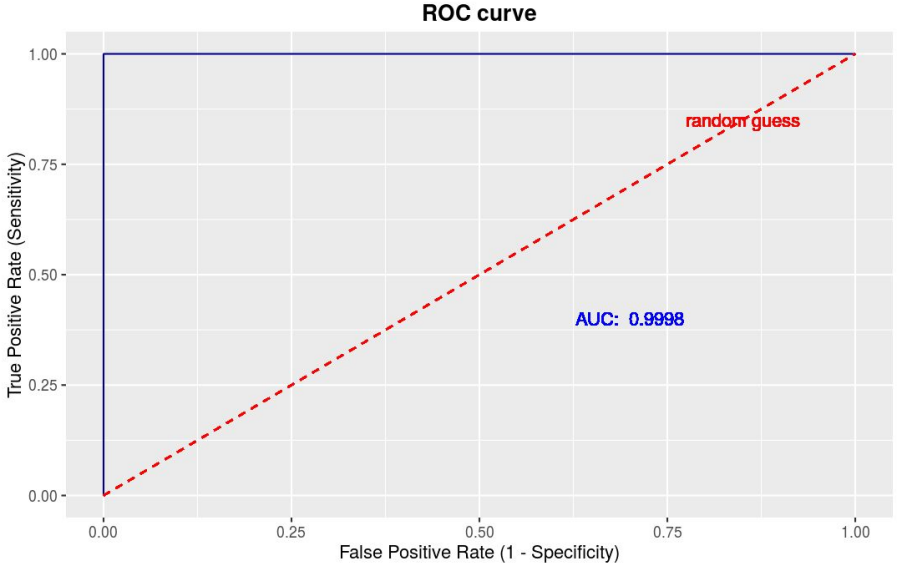



A **random classifier** has an **AUC ROC of 0.5**

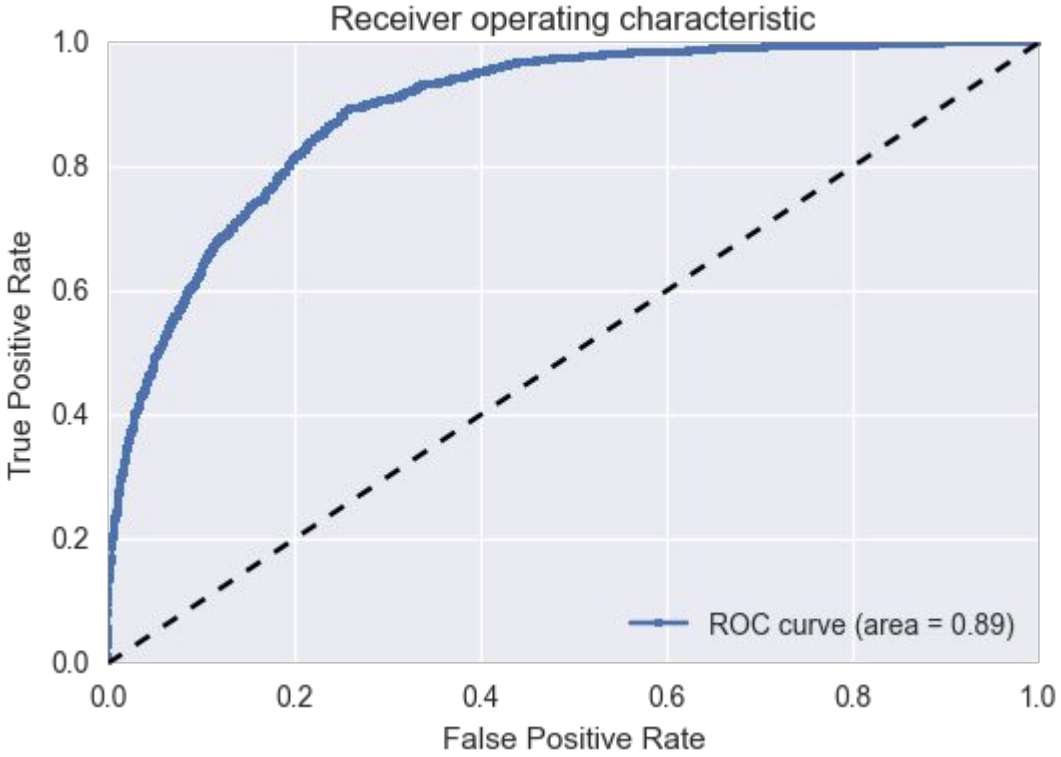
# ROC of a perfect classifier



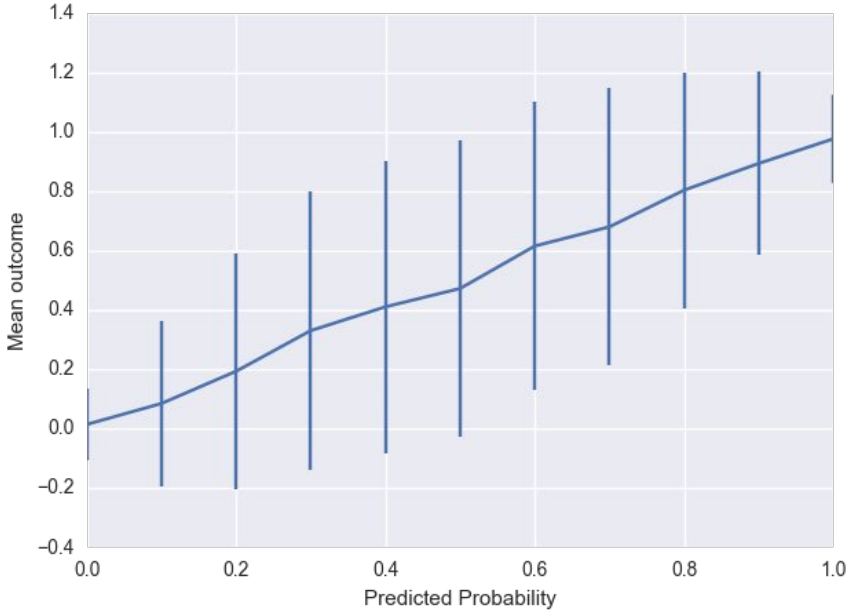

A **perfect classifier** has an **AUC ROC of 1**

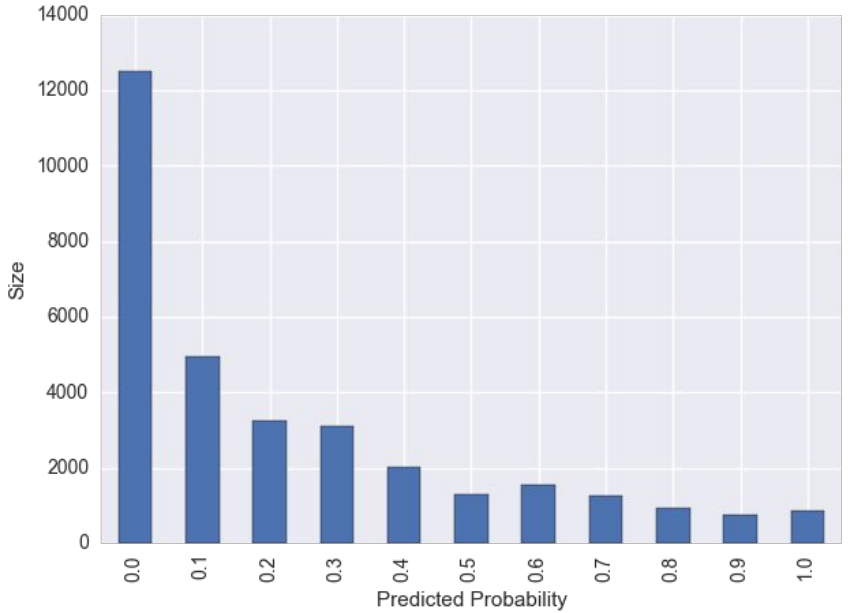
# Predicting income over $50k




- In the notebook example on Github, we find the AUC ROC is 0.9 which is very good.
- In general, anything over 0.7 is good, anything over 0.8 is amazing, and anything over 0.9 is suspicious (more on this later).

# Accuracy as a function of propensity




- These plots are easily understood by non-technical people.
- Also useful when you run experiments to understand the impact of an action on people with a particular propensity.

# Optimizing your model

Regularization and feature selection

# Model Optimization

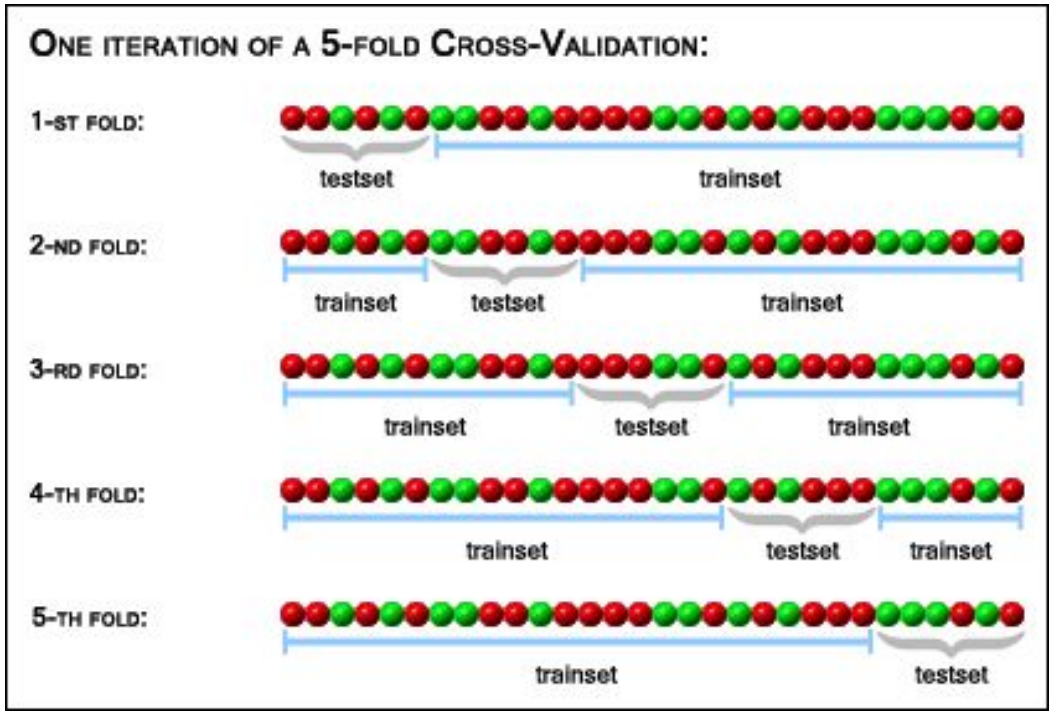
- Just as with regression, we risk over fitting our model if we include too many variables.
- We also risk instability and lack of confidence in variables if we have collinear features.
- The exact same methods are used to optimize models in classification problems, but our metric for evaluating performance is now AUC ROC and not R^2.

# Regularization for Classification

$$\min_{\beta} -\sum_{k=1}^{n} \log p(y_i|x_i, \beta) + \lambda \sum_{m=1}^{M} |\beta_m|^p$$

- As before, we want to find the lambda which results in the best performance on test data.
- Here though we will use AUC ROC as our performance metric.

# Cross Validation



# Example - Income Data Set

Question: Does the individual make over $50k? (yes/no)

Variables:

- **age:** continuous.
- **workclass:** Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **education:** Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- **education-num:** continuous.
- **marital-status:** Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation:** Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- **relationship:** Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race:** White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex:** Female, Male.
- **capital-gain:** continuous.
- **capital-loss:** continuous.
- **hours-per-week:** continuous.
- **native-country:**

# Outline

1. Inspect the data, convert categorical variables to new features. Convert string based variables to numerical values.

2. Train the model using Logistic Regression.

3. Evaluate performance using AUC ROC and accuracy vs percentile.

4. Check for collinear feature and plot the correlation matrix.

5. Optimize by using L1 and L2 regularization (measuring performance on testing data).

# A look at the data

```
In [272]: df.head()
```

Out[272]:

| | Age | Workclass | Sector | Education | Education-num | Marital-Status | Occupation | Relationship | Race | Sex | Capital-Gain | Capital-Loss | Hours-Per-Week | Native-Country | y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 39 | State-gov | 77516 | Bachelors | 13 | Never-married | Adm-clerical | Not-in-family | White | Male | 2174 | 0 | 40 | United-States | <=50K |
| **1** | 50 | Self-emp-not-inc | 83311 | Bachelors | 13 | Married-civ-spouse | Exec-managerial | Husband | White | Male | 0 | 0 | 13 | United-States | <=50K |
| **2** | 38 | Private | 215646 | HS-grad | 9 | Divorced | Handlers-cleaners | Not-in-family | White | Male | 0 | 0 | 40 | United-States | <=50K |
| **3** | 53 | Private | 234721 | 11th | 7 | Married-civ-spouse | Handlers-cleaners | Husband | Black | Male | 0 | 0 | 40 | United-States | <=50K |
| **4** | 28 | Private | 338409 | Bachelors | 13 | Married-civ-spouse | Prof-specialty | Wife | Black | Female | 0 | 0 | 40 | Cuba | <=50K |

# How do we deal with categorical data?

| ıip | Race | Sex | Capital-Gain |
|---|---|---|---|
| ily | White | Male | 2174 |
| | White | Male | 0 |
| ily | White | Male | 0 |
| | Black | Male | 0 |
| | Black | Female | 0 |

- Notice two issues we encounter here:
  - Many of the variables are strings.
  - How can we input categorical features into a model?

$$y = \beta \cdot \text{male}$$ makes no sense!

# How do we deal with categorical data?

**Solution:** Create dummy variables. This is also known as one-hot encoding.

```
In [339]: df_workclass=pd.get_dummies(df['Workclass'])
          df_sector=pd.get_dummies(df['Sector'])
          df_education=pd.get_dummies(df['Education'])
          df_occupation=pd.get_dummies(df['Occupation'])
          df_relationship = pd.get_dummies(df['Relationship'])
          df_race = pd.get_dummies(df['Sex'])
          df_country=pd.get_dummies(df['Native-Country'])

In [340]: df_final = pd.concat([df[['Age','Capital-Gain','Capital-Loss']],df_workclass,df_education,df_relationship,df_race,df_cou
```

For each category (ie. Gender: MALE, FEMALE), we create a new column. The MALE column will be 1 when the person is male, 0 otherwise.

# How do we deal with categorical data?

**Solution:** Create dummy variables. This is also known as one hot encoding.

```
In [565]: df['Sex'].head()

Out[565]: 0       Male
          1       Male
          2       Male
          3       Male
          4     Female
          Name: Sex, dtype: object
```

```
In [563]: df_final[[u' Female',u' Male']]
```

Out[563]:

| | Female | Male |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 0 | 1 |
| **2** | 0 | 1 |
| **3** | 0 | 1 |
| **4** | 1 | 0 |

# Convert the outcome variable

We convert the $y$ variable into a binary outcome (do they make over 50k or not)

In [279]:
```
def get_y(y):
    if y.find("<=")>-1:
        return 0
    else:
        return 1
```

Now that the **data is processed,** we can **train a model** and **evaluate its performance.**

In [9]:
```
# X are features and y is predictive variable - cancelled HD or not.
X = df_final
y = df['y'].apply(lambda y : get_y(y))
```

# Train model and compute AUC ROC

1. Split data into **testing** and **training**.

2. Train model using sklearn's **LogisticRegression** on the **training data**.

3. Make **predictions** on the **testing data.**

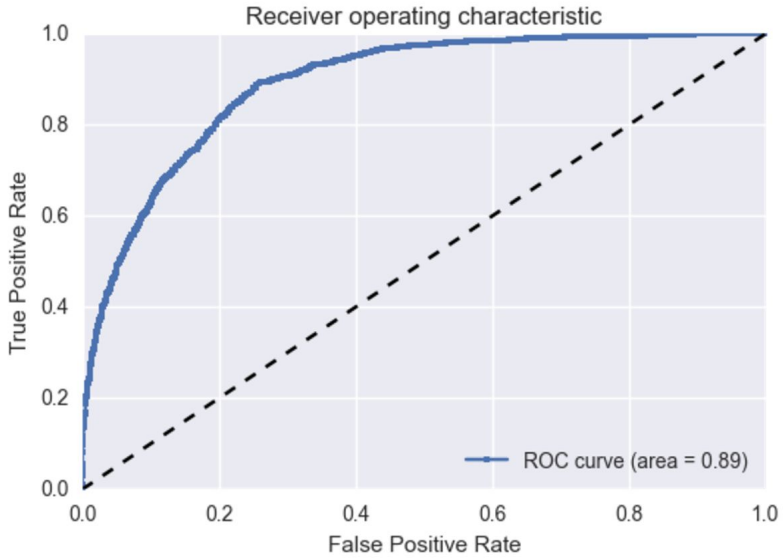
4. Evaluate performance using AUC ROC.

# Train model and compute AUC ROC

```
# Computes the AUC ROC of the classifier.
def generate_auc(X,y,clf_class,**kwargs):
    # Construct a kfolds object
    random_state = np.random.RandomState(0)
    # Construct training and testing set.
    X, y = shuffle(X, y, random_state=random_state)
    n_samples, n_features = X.shape
    half = int(n_samples/1.2)
    X_train, X_test = X[:half], X[half:]
    y_train, y_test = y[:half], y[half:]

    # Choose SVC classifier.
    classifier = clf_class(**kwargs)
    probas_ = classifier.fit(X_train, y_train).predict_proba(X_test)

    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    roc_auc = auc(fpr, tpr)
    print "Area under the ROC curve : %f" % roc_auc
    return fpr, tpr, roc_auc, thresholds
```

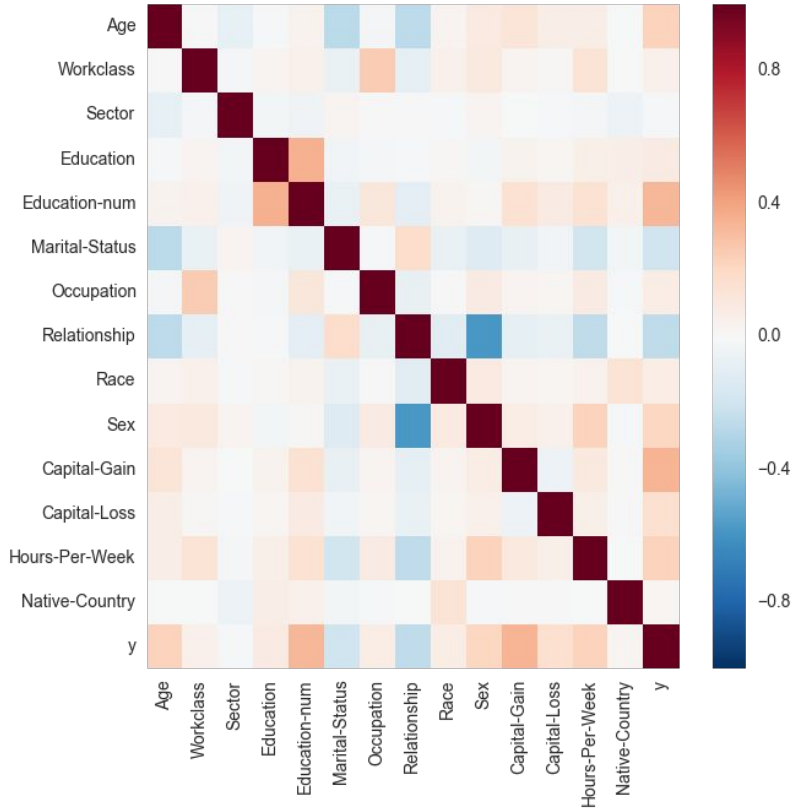
# Train model and compute AUC ROC

```
Area under the ROC curve : 0.891329
```




- Our first model seems pretty good! An **AUC ROC of 0.9.**
- I often just feed everything into a classifier/regression algorithm before I begin optimizing and analyzing the data further.
- Next, we will look at correlation between features, followed by optimizing the complexity of the model.

# Correlation of features



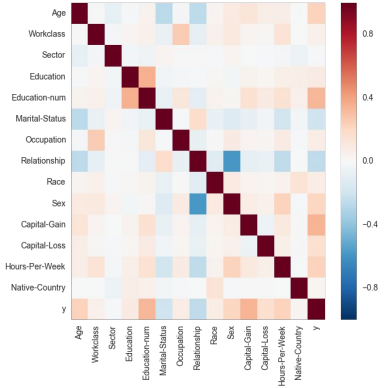

**Positive Correlations:**

- Capital-Gain and y
- Education-num and y
- Workclass and Occupation
- Education-num and Education

**Negative Correlations:**

- Sex and Relationship
- Age and Relationship
- Marital status and Age

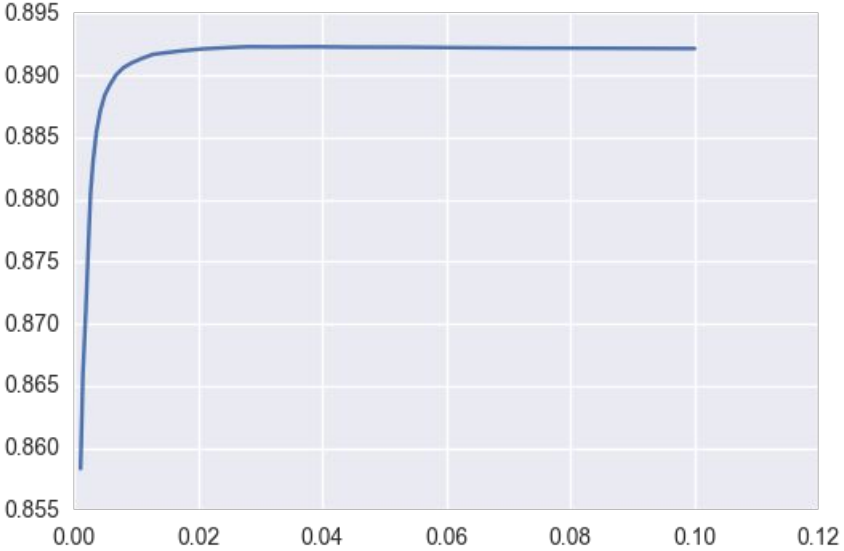
# Correlation of features - Removing Them



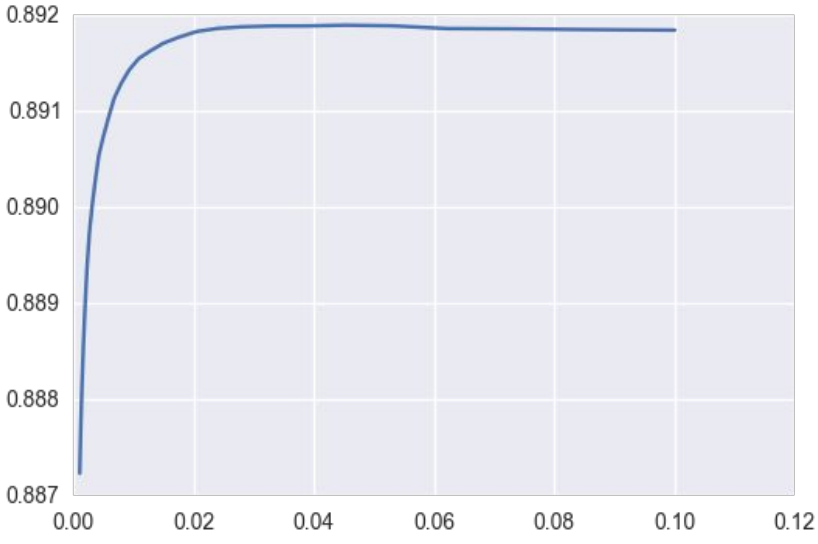

**How to remove correlated features:**

- L1 and L2 regularization are the standard methods.
  (how we will do this)
- PCA (to be covered soon) also works for this purpose.
- Old School (removing features with large p values manually).

# Lasso vs Ridge regression



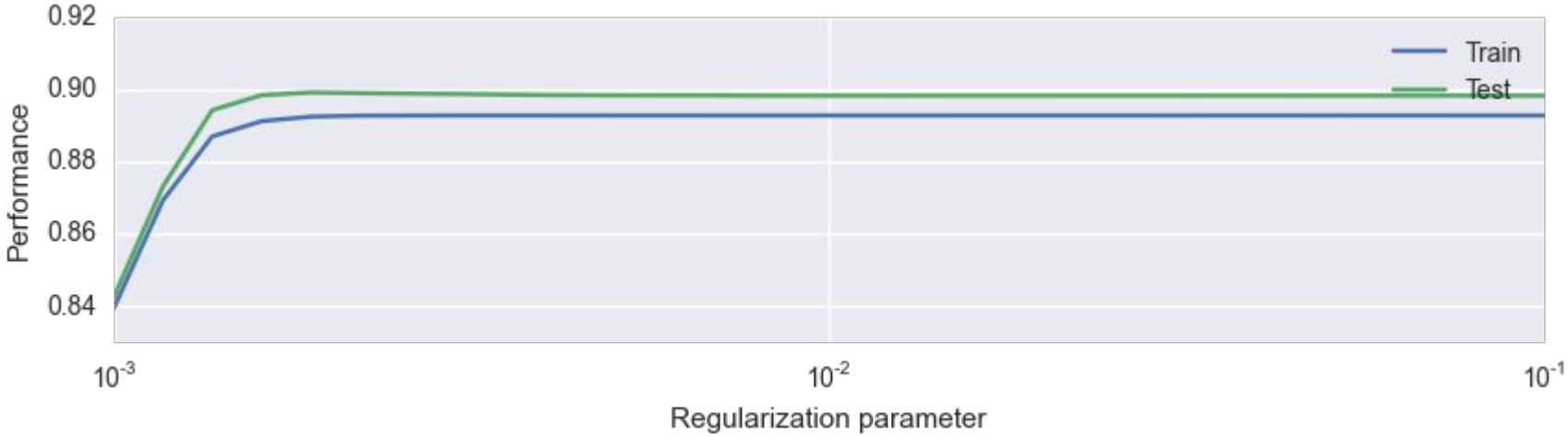

$p = 1$




$p = 2$

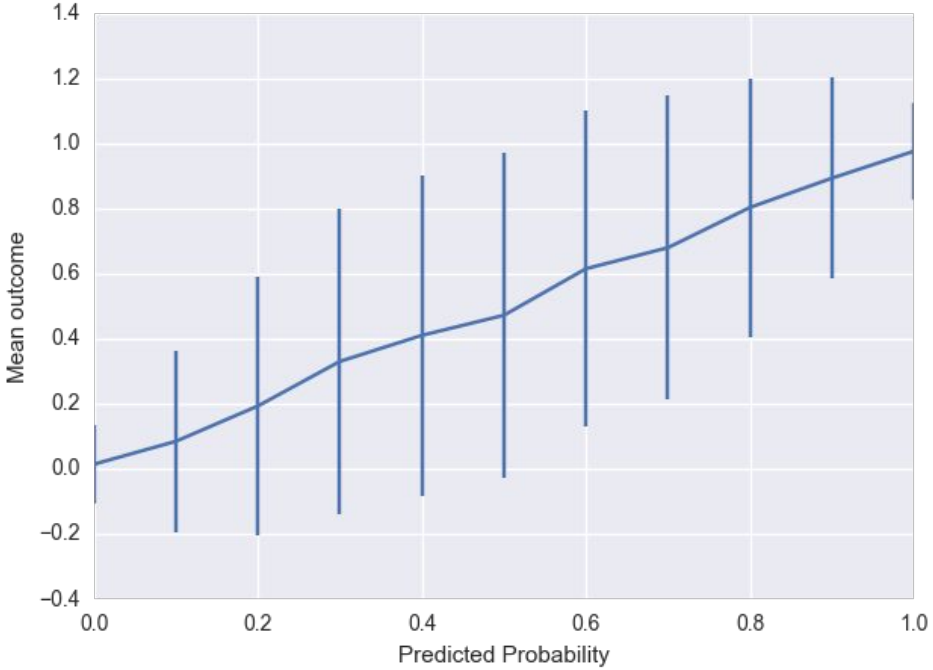
$$\min_{\beta} -\sum_{k=1}^{n} \log p(y_i|x_i, \beta) + \lambda \sum_{m=1}^{M} |\beta_m|^p$$

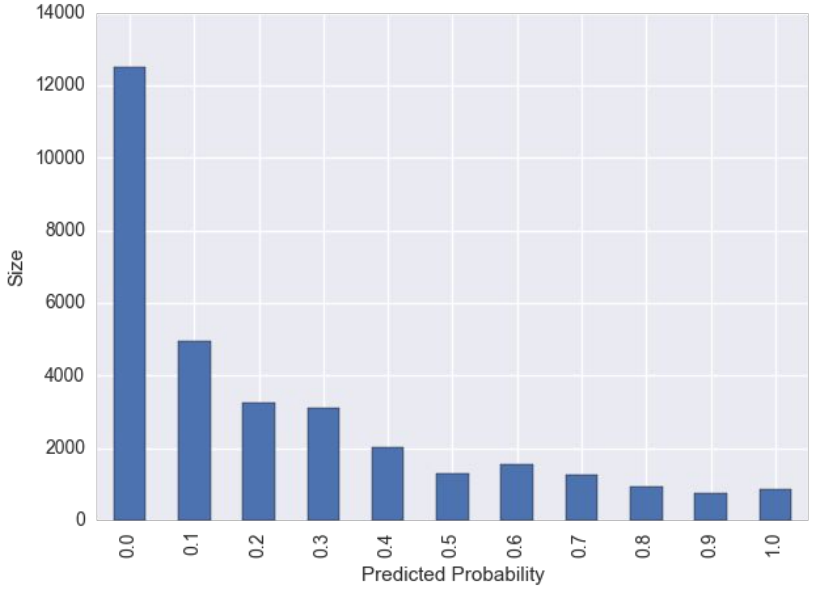
# Optimizing Continued




$$\min_{\beta} -\sum_{k=1}^{n} \log p(y_i|x_i,\beta) + \lambda \sum_{m=1}^{M} |\beta_m|^p$$

# Accuracy vs Percentile



# Feature Importance

Regularization and feature selection

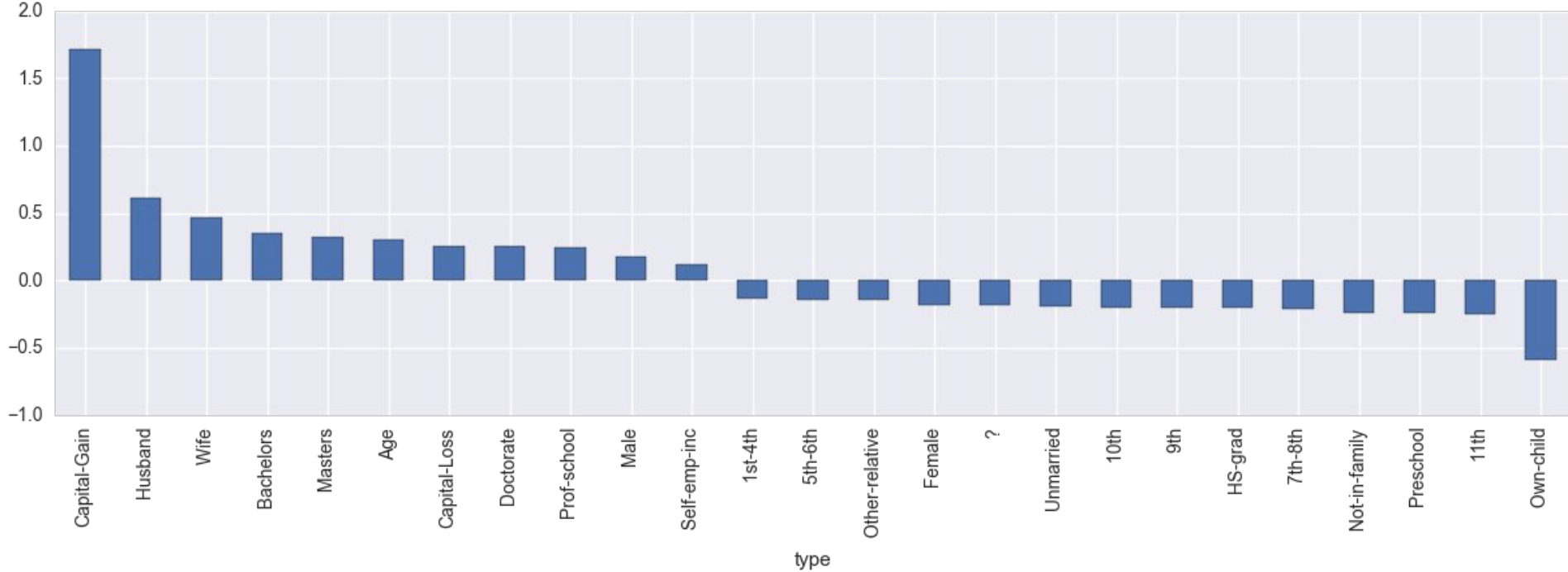
# Fit the optimal model

In [304]:
```
alpha = alphas[np.argmax(scores)]
```

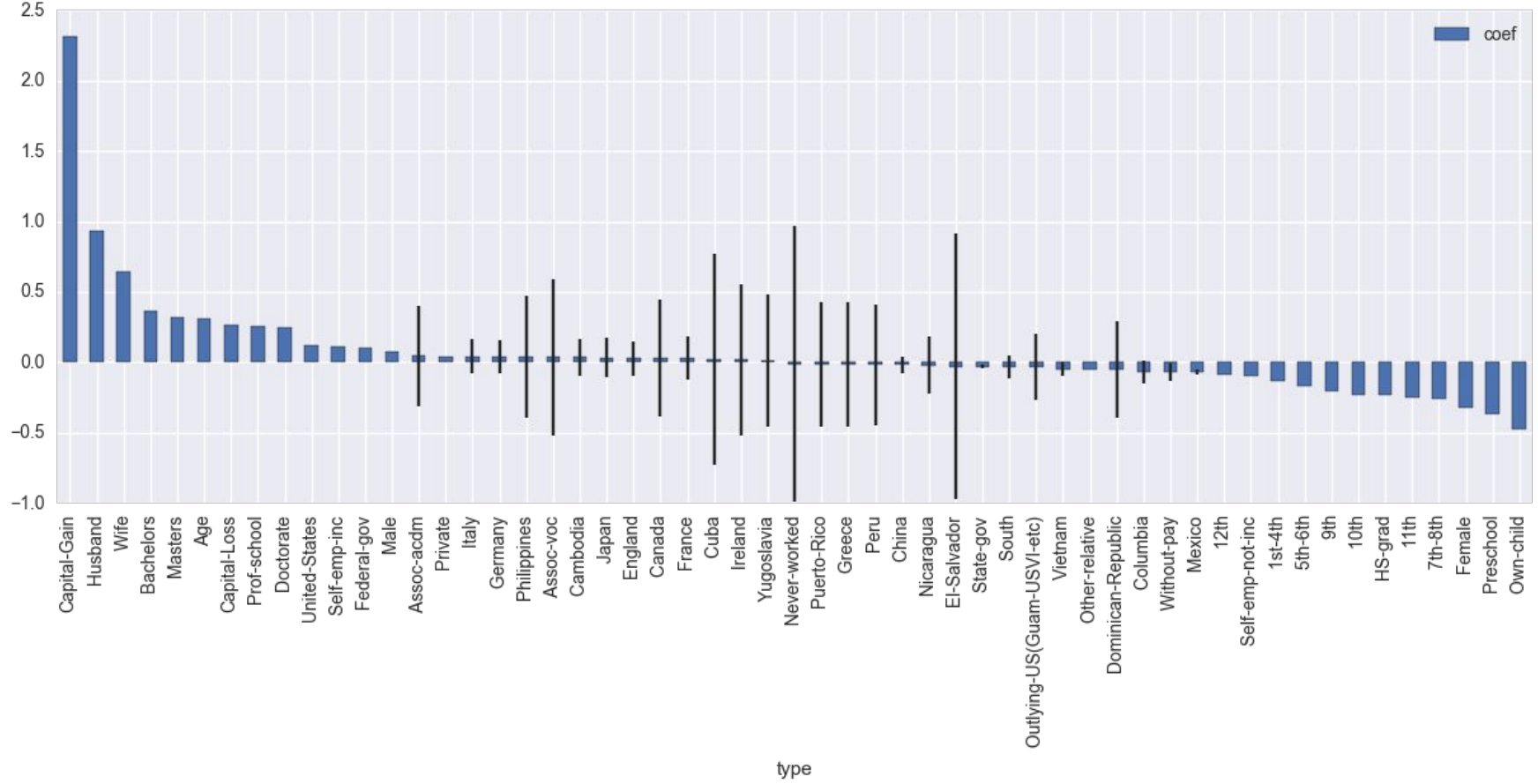
In [356]:
```
regr = LogisticRegression(C=alpha,penalty='l1')

# Train the model using the training sets
regr.fit(X_train, y_train)
```

Out[356]:
```
LogisticRegression(C=0.10000000000000001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

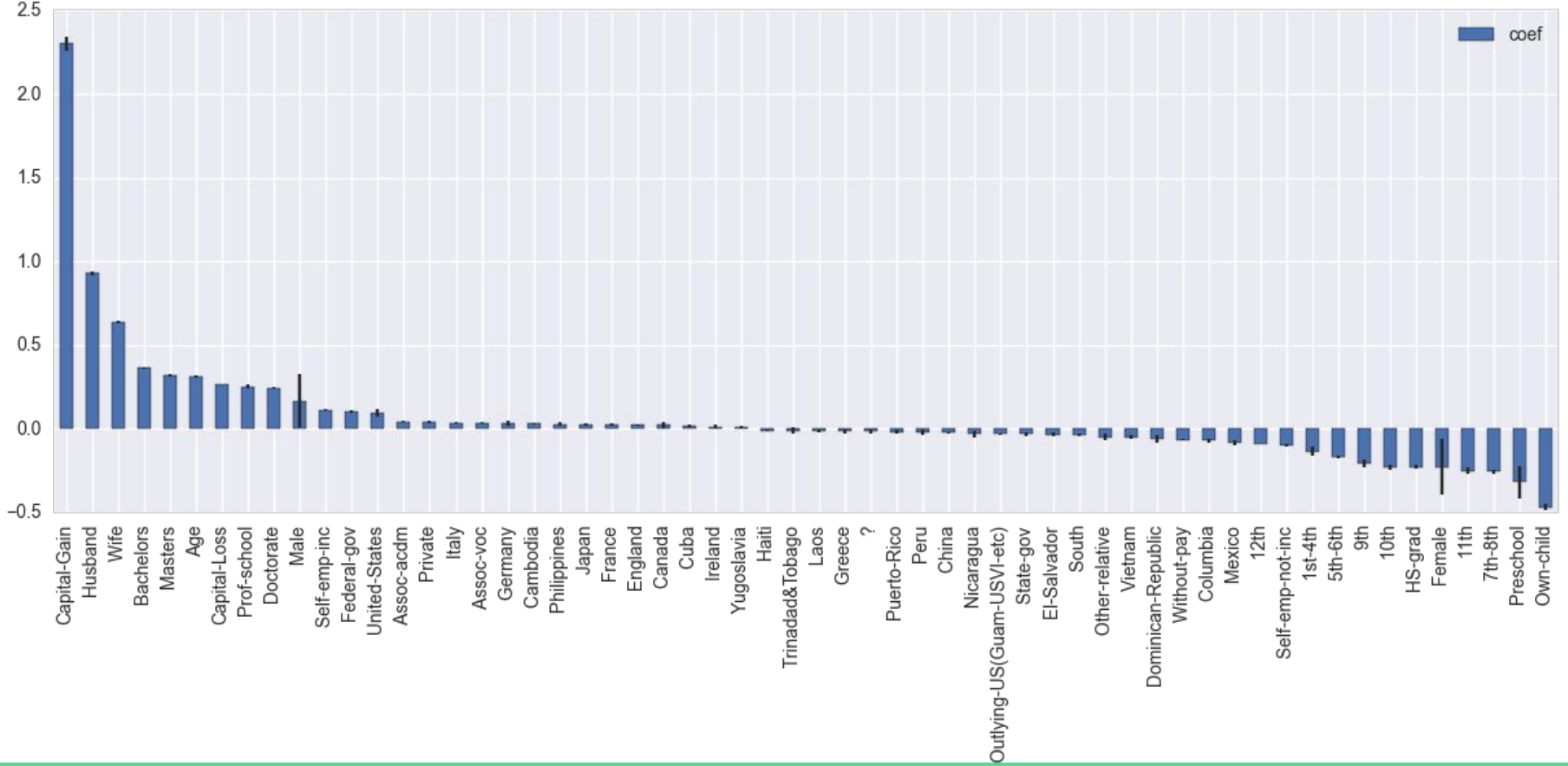
# Sort and plot coefficients



# P Values


coef
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
Capital-Gain
Husband
Wife
Bachelors
Masters
Age
Capital-Loss
Prof-school
Doctorate
United-States
Self-emp-inc
Federal-gov
Male
Assoc-acdm
Private
Italy
Germany
Philippines
Assoc-voc
Cambodia
Japan
England
Canada
France
Cuba
Ireland
Yugoslavia
Never-worked
Puerto-Rico
Greece
Peru
China
Nicaragua
El-Salvador
State-gov
South
Outlying-US(Guam-USVI-etc)
Vietnam
Other-relative
Dominican-Republic
Columbia
Without-pay
Mexico
12th
Self-emp-not-inc
1st-4th
5th-6th
9th
10th
HS-grad
11th
7th-8th
Female
Preschool
Own-child
type

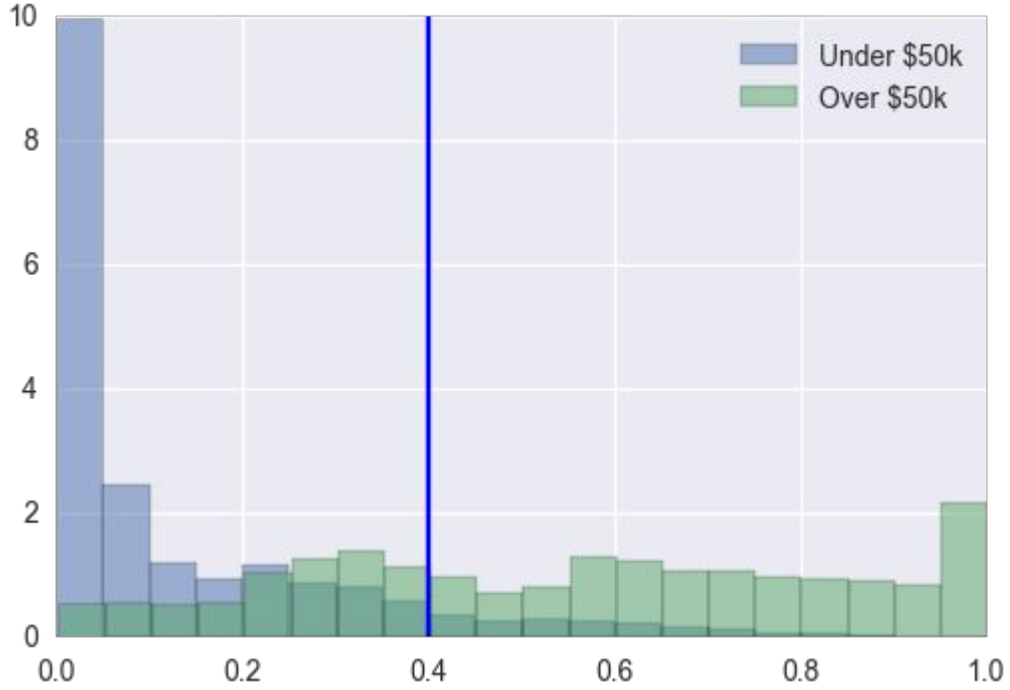
# Cross Validation



# Choosing a cutoff

What is the best cutoff for our model?

# What’s the best cut off point?



# The best cut off


$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

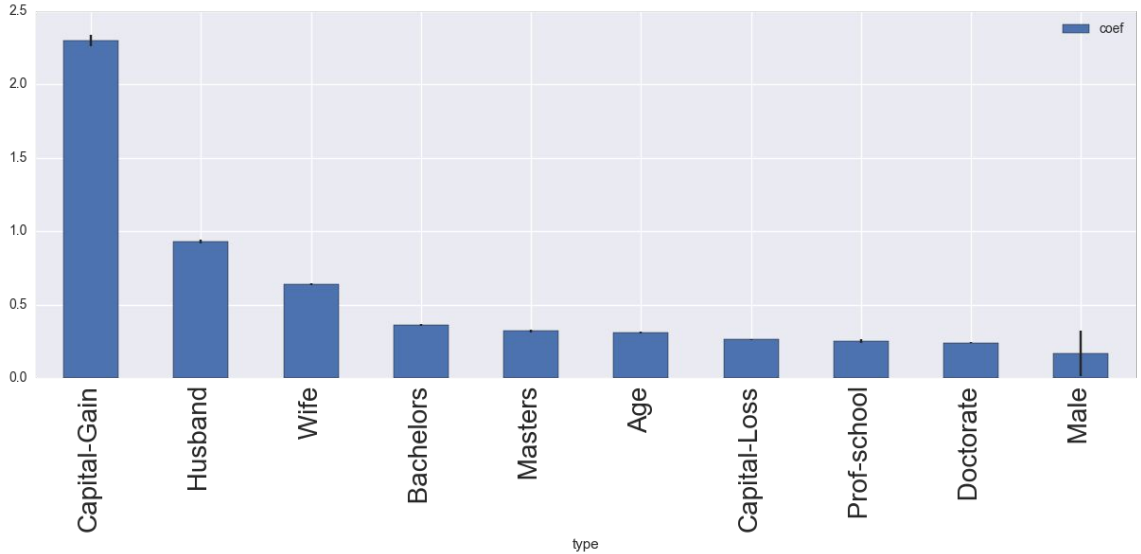
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

**We want TPR to be large and FPR to be small.**

# Our final model

$$p_\beta(\mathbf{x}) = \frac{1}{1 + \exp(-\beta^T \cdot \mathbf{x})}$$

$$\beta = [2.3, 0.93, 0.64, 0.36, 0.32, 0.31, 0.27, 0.25, 0.24, 0.17]$$



# How do we evaluate a model? Appendix

| | | Predicted condition | | | |
|---|---|---|---|---|---|
| | Total population | Predicted Condition positive | Predicted Condition negative | Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$ | |
| True condition | condition positive | **True positive** | **False Negative** (Type II error) | True positive rate (TPR), Sensitivity, Recall, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$ | False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$ |
| | condition negative | **False Positive** (Type I error) | **True negative** | False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$ | True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$ |
| | Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$ | Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$ | False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$ | Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$ | Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$ |
| | | False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$ | Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$ | Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$ | |