Logistic Regression

Thomas Schwarz

Categorical Data

- Outcomes can be categorical
  - Often, outcome is binary:
    - President gets re-elected or not
    - Customer is satisfied or not
  - Often, explanatory variables are categorical as well
    - Person comes from an under-performing school
    - Order was made on a week-end
    - …

Prediction Models for Binary Outcomes

- Famous example:
  - Taken an image of a pet, predict whether this is a cat or a dog

Prediction Models for Binary Outcomes

- Bayes: ***generative classifier***
  - Predicts indirectly $P(c \mid d)$
- $\hat{c} =$ arg max$_{c \in C} P(d \mid c) P(c)$

  Likelihood Prior

  - Evaluates product of likelihood and prior
    - Prior: Probability of a category c without looking at data
    - Likelihood: Probability of observing data if from a category c

Prediction Models for Binary Outcomes

- Regression is a ***discriminative classifier***
  - Tries to learn directly the classification from data
    - E.g.: All dog pictures have a collar
      - Collar present —> predict dog
      - Collar not present —> predict cat
  - Computes directly $P(c \mid d)$

Prediction Models for Binary Outcomes

- Regression:
  - Supervised learning: Have a training set with classification provided
  - Input is given as vectors of numerical features
    - $\mathbf{x}^{(i)} = (x_{1,i}, x_{2,i}, \ldots, x_{n,i})$
  - Classification function that calculates the predicted class $\hat{y}(\mathbf{x})$
  - An objective function for learning: Measures the goodness of fit between true outcome and predicted outcome
  - An algorithm to optimize the objective function

Prediction Models for Binary Outcomes

- Linear Regression:
  - Classification function of type
    - $\hat{y}\left((x_1, x_2, \ldots, x_n)\right) = a_1 x_1 + a_2 x_2 + \ldots a_n x_n + b$
  - Objective function (a.k.a cost function)
    - Sum of squared differences between predicted and observed outcomes
      - E.g. Test Set $T = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(m)}\}$
      - Minimize cost function $\sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$

Prediction Models for Binary Outcomes

- Linear regression can predict a numerical value
    - It can be made to predict a binary value
        - If the predictor is higher than a cut-off value: predict yes
        - Else predict no
- But there are better ways to generate a binary classifier

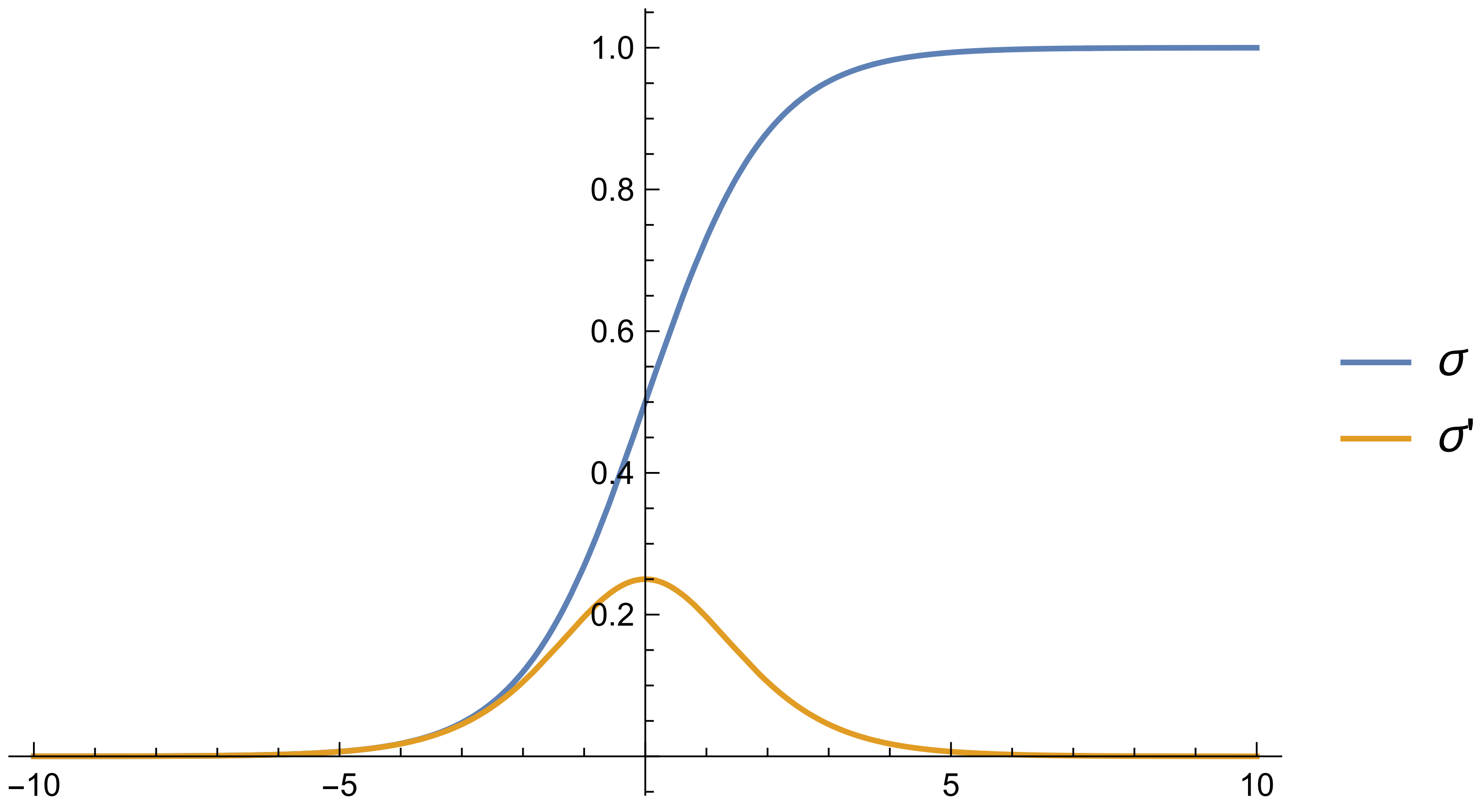
Prediction Models for Binary Outcomes

- Good binary classifier:
  - Since we want to predict the probability of a category based on the features:
    - Should look like a probability
  - Since we want to optimize:
    - Should be easy to differentiate
  - Best candidate classifier that has emerged:
    - Sigmoid classifier

Logistic Regression

- Use logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



Logistic Regression

- Combine with linear regression to obtain logistic regression approach:
  - Learn best weights in
    - $\hat{y}\left((x_1, x_2, \ldots, x_n)\right) = \sigma(b + w_1x_1 + w_2x_2 + \ldots w_nx_n)$
  - We know interpret this as a probability for the positive outcome '+'
  - Set a ***decision boundary*** at 0.5
    - This is no restriction since we can adjust b and the weights

Logistic Regression

- We need to measure how far a prediction is from the true value
  - Our predictions $\hat{y}$ and the true value y can only be 0 or 1
    - If $y = 1$: Want to support $\hat{y} = 1$ and penalize $\hat{y} = 0$.
    - If $y = 0$: Want to support $\hat{y} = 0$ and penalize $\hat{y} = 1$.
  - One successful approach:
    - Loss$(\hat{y}, y) = \hat{y}^y(1 - \hat{y})^{(1-y)}$

Logistic Regression

- Easier: Take the negative logarithm of the loss function
  - Cross Entropy Loss

$$L_{CE} = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Logistic Regression

- This approach is successful, because we can use Gradient Descent
  - Training set of size m
  - Minimize $\sum_{i=1}^{m} \mathrm{L_{CE}}(y^{(i)}, \hat{y}^{(i)})$
  - Turns out to be a convex function, so minimization is simple! (As far as those things go)
  - Recall:

$$\hat{y}\left((x_1, x_2, \ldots, x_n)\right) = \sigma(b + w_1 x_1 + w_2 x_2 + \ldots w_n x_n)$$

  - We minimize with respect to the weights and b

Logistic Regression

- Calculus:

$$\frac{\delta \mathrm{L}_{\mathrm{CE}}(w, b)}{\delta w_j} = \left(\sigma(w_1 x_1 + \ldots w_n x_n + b) - y\right) x_j$$

$$= (\hat{y} - y) x_j$$

- Difference between true y and estimated outcome $\hat{y}$, multiplied by input coordinate

Logistic Regression

- Stochastic Gradient Descent
    - Until gradient is almost zero:
        - For each training point $x^{(i)}, y^{(i)}$:
            - Compute prediction $\hat{y}^{(i)}$
            - Compute loss
            - Compute gradient
            - Nudge weights in the opposite direction using a learning weight η
                - $(w_1, \ldots, w_n) \leftarrow (w_1, \ldots, w_n) - \eta \nabla \mathrm{L}_{\mathrm{CE}}$
            - Adjust η

Logistic Regression

- Stochastic gradient descent uses a single data point
    - Better results with random batches of points at the same time

Lasso and Ridge Regression

- If the feature vector is long, danger of overfitting is high
  - We learn the details of the training set
    - Want to limit the number of features with positive weight
  - Dealt with by adding a regularization term to the cost function
    - Regularization term depends on the weights
      - Penalizes large weights

Lasso and Ridge Regression

- L2 regularization:
  - Use a quadratic function of the weights
    - Such as the euclidean norm of the weights
  - Called ***Ridge Regression***
    - Easier to optimize

Lasso and Ridge Regression

- L1 regularization
  - Regularization term is the sum of the absolute values of weights
  - Not differentiable, so optimization is more difficult
  - BUT: effective at lowering the number of non-zero weights
- Feature selection:
  - Restrict the number of features in a model
  - Usually gives better predictions

Examples

- Example: quality.csv
  - Try to predict whether patient labeled care they received as poor or good

quality

MemberID	InpatientDays	ERVisits	OfficeVisits	Narcotics	DaysSinceLastERVisit	Pain	TotalVisits	ProviderCount	MedicalClaims	ClaimLines	StartedOnCombination	AcuteDrugGapSmall	PoorCare
1	0	0	18	1	731	10	18	21	93	222	FALSE	0	0
2	1	1	6	1	411	0	8	27	19	115	FALSE	1	0
3	0	0	5	3	731	10	5	16	27	148	FALSE	5	0
4	0	1	19	0	158	34	20	14	59	242	FALSE	0	0
5	8	2	19	3	449	10	29	24	51	204	FALSE	0	0
6	2	0	9	2	731	6	11	40	53	156	FALSE	4	1
7	16	1	8	1	173.9583333	4	25	19	40	261	FALSE	0	0
8	2	0	8	0	731	5	10	11	28	87	FALSE	0	0
9	2	1	4	3	45	5	7	28	20	98	FALSE	0	1
10	4	2	0	2	104	2	6	21	17	66	FALSE	0	0
11	6	5	20	2	156	9	31	19	43	126	FALSE	2	0
12	0	0	7	4	731	0	7	8	23	41	FALSE	2	0
13	0	1	3	1	389	23	4	13	18	70	FALSE	0	0
14	1	1	20	3	594.9583333	16	22	18	48	133	FALSE	0	0
15	6	2	31	3	640.9583333	70	39	28	101	233	FALSE	0	0
16	0	0	8	0	731	0	8	5	19	48	FALSE	0	0
17	2	0	9	0	731	29	11	22	39	120	FALSE	0	0
18	3	0	20	1	731	13	23	17	34	73	FALSE	3	1
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Examples

- First column is an arbitrary patient ID
  - we make this the index
- One column is a Boolean, when imported into Python
  - so we change it to a numeric value

```
df = pd.read_csv('quality.csv', sep=',', index_col=0)
df.replace({False:0, True:1}, inplace=True)
```

Examples

- Farmington Heart Data Project:
  - https://framinghamheartstudy.org
  - Monitoring health data since 1948
  - 2002 enrolled grandchildren of first study

Examples

framingham

male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
1	39	4	0	0	0	0	0	0	195	106	70	26.97	80	77	0
0	46	2	0	0	0	0	0	0	250	121	81	28.73	95	76	0
1	48	1	1	20	0	0	0	0	245	127.5	80	25.34	75	70	0
0	61	3	1	30	0	0	1	0	225	150	95	28.58	65	103	1
0	46	3	1	23	0	0	0	0	285	130	84	23.1	85	85	0
0	43	2	0	0	0	0	1	0	228	180	110	30.3	77	99	0
0	63	1	0	0	0	0	0	0	205	138	71	33.11	60	85	1
0	45	2	1	20	0	0	0	0	313	100	71	21.68	79	78	0
1	52	1	0	0	0	0	1	0	260	141.5	89	26.36	76	79	0
1	43	1	1	30	0	0	1	0	225	162	107	23.61	93	88	0
0	50	1	0	0	0	0	0	0	254	133	76	22.91	75	76	0

Examples

- Contains a few NaN data
  - We just drop them

```
df = pd.read_csv('framingham.csv', sep=',')
df.dropna(inplace=True)
```

Logistic Regression in Stats-Models

- Import statsmodels.api

```
import statsmodels.api as sm
```

- Interactively select the columns that gives us high p-values

```
cols = [ 'Pain', 'TotalVisits',
         'ProviderCount',
         'MedicalClaims', 'ClaimLines',
         'StartedOnCombination',
         'AcuteDrugGapSmall',]
```

Logistic Regression in Stats-Models

- Create a logit model
  - Can do as we did for linear regression with a string
  - Can do using a dataframe syntax

```
logit_model=sm.Logit(df.PoorCare,df[cols])
result=logit_model.fit()
```

  - Print the summary pages

```
print(result.summary2())
```

Logistic Regression in Stats-Models

- Print the results
    - `print(result.pred_table())`
- This gives the "confusion matrix"
    - Coefficient [i,j] gives:
        - predicted i values
        - actual j values

Actual Values

Predicted Values	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

Logistic Regression in Stats-Models

- Quality prediction:
  - ```
    [[91.  7.]
     [18. 15.]]
    ```
- 7 False negative and 18 false positives
    ```

Logistic Regression in Stats-Models

- Heart Event Prediction:
  - ```
    [[3075.   26.]
     [ 523.   34.]]
    ```
- 26 false negatives
- 523 false positives
    ```

Logistic Regression in Stats-Models

- Can try to improve using Lasso

```
result=logit_model.fit_regularized()
```

Logistic Regression in Stats-Models

- Can try to improve selecting only columns with high P-values

```
Optimization terminated successfully.
         Current function value: 0.423769
         Iterations 6
                         Results: Logit
================================================================
Model:              Logit            Pseudo R-squared: 0.007
Dependent Variable: TenYearCHD       AIC:              3114.2927
Date:               2020-07-12 18:18 BIC:              3157.7254
No. Observations:   3658             Log-Likelihood:   -1550.1
Df Model:           6                LL-Null:          -1560.6
Df Residuals:       3651             LLR p-value:      0.0019166
Converged:          1.0000           Scale:            1.0000
No. Iterations:     6.0000
----------------------------------------------------------------
                  Coef.  Std.Err.    z    P>|z|   [0.025  0.975]
----------------------------------------------------------------
currentSmoker     0.0390   0.0908  0.4291 0.6679 -0.1391  0.2170
BPMeds            0.5145   0.2200  2.3388 0.0193  0.0833  0.9457
prevalentStroke   0.7716   0.4708  1.6390 0.1012 -0.1511  1.6944
prevalentHyp      0.8892   0.0983  9.0439 0.0000  0.6965  1.0818
diabetes          1.4746   0.2696  5.4688 0.0000  0.9461  2.0030
totChol          -0.0067   0.0007 -9.7668 0.0000 -0.0081 -0.0054
glucose          -0.0061   0.0019 -3.2113 0.0013 -0.0098 -0.0024
================================================================
```

Logistic Regression in Stats-Models

- Select the columns
  - ```
    cols = ['currentSmoker', 'BPMeds',
            'prevalentStroke', 'prevalentHyp',
            'diabetes', 'totChol','glucose']
    ```
- Get a better (?) confusion matrix:
  - ```
     [[3086.   15.]
      [ 549.    8.]]
    ```

  - False negatives has gone down
  - False positives has gone up

Logistic Regression in Scikit-learn

- Import from sklearn

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

Logistic Regression in Scikit-learn

- Create a logistic regression object and fit it on the data

```
logreg = LogisticRegression()
logreg.fit(X=df[cols], y = df.TenYearCHD)
y_pred = logreg.predict(df[cols])
confusion_matrix = confusion_matrix(df.TenYearCHD,
y_pred)
print(confusion_matrix)
```

Logistic Regression in Scikit-learn

- Scikit-learn uses a different algorithm
  - Confusion matrix on the whole set is
    -

```
[[3087   14]
 [ 535   22]]
```

Logistic Regression in Scikit-learn

- Can also divide the set in training and test set

```
X_train, X_test, y_train, y_test =
   train_test_split(df[cols],
                    df.TenYearCHD,
                    test_size=0.3,
                    random_state=0)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

Logistic Regression in Scikit-learn

- Confusion matrix
  - ```
    [[915   1]
     [176   6]]
    ```
    ```

Measuring Success

	precision	recall	f1-score	support
0	0.84	1.00	0.91	916
1	0.86	0.03	0.06	182
accuracy			0.84	1098
macro avg	0.85	0.52	0.49	1098
weighted avg	0.84	0.84	0.77	1098

Measuring Success

- How can we measure accuracy?
  - accuracy = (fp+fn)/(tp+tn+fp+fn)
    - Unfortunately, because of skewed data sets, often very high
  - precision = tp/(tp+fp)
  - recall = tp/(tp+fn)
  - F measure = harmonic mean of precision and recall

Probit Regression

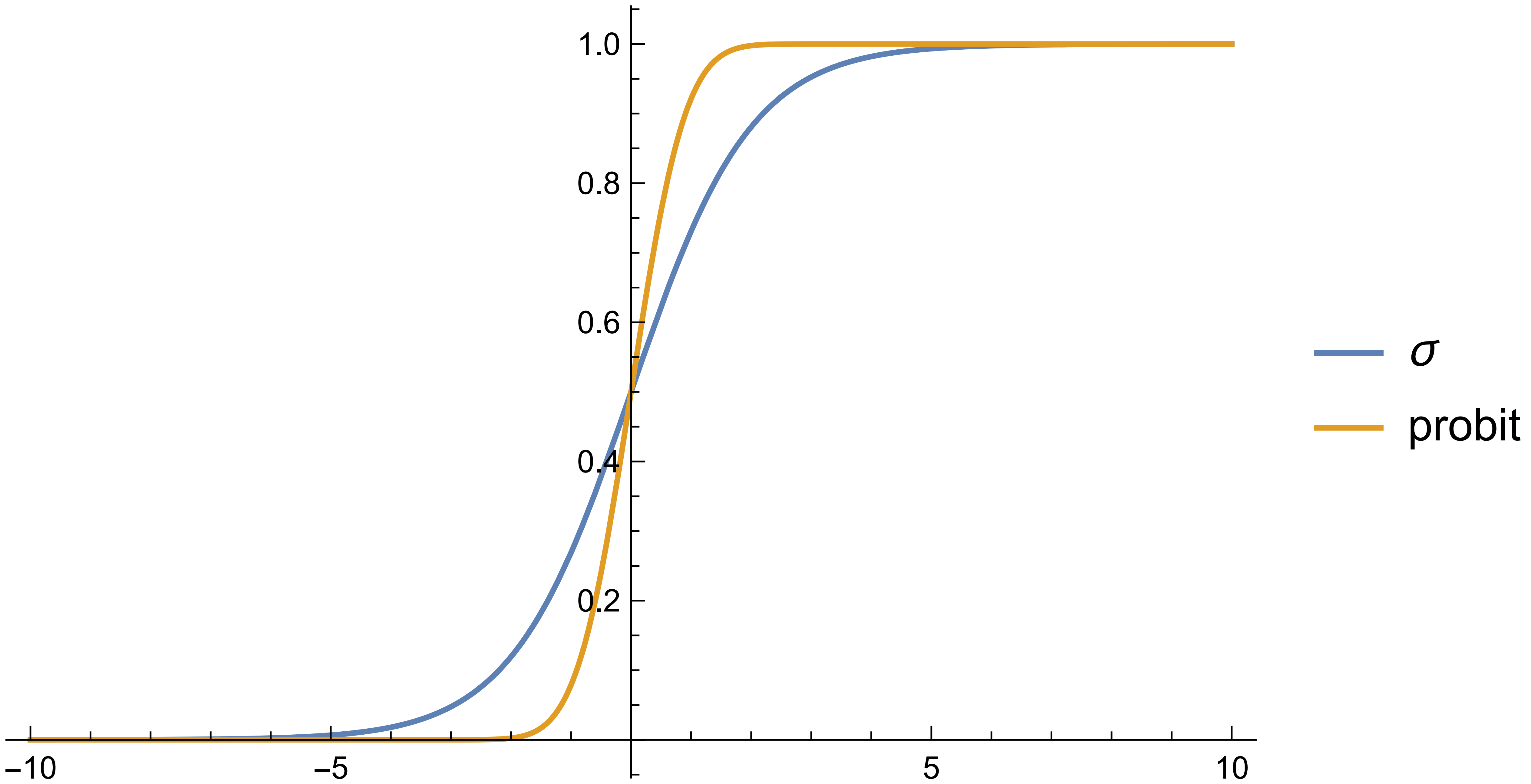
- Instead of using the logistic function σ, can also use the cumulative distribution function of the normal distribution

$$\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2) dt$$

- Predictor is then

$$\frac{1}{2}\left(1 + \text{erf}(b + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n)\right)$$

Probit Regression



Probit Regression

- Calculations with probit are more involved
  - Statsmodels implements it
    - `from statsmodels.discrete.discrete_model import Probit`
  - Fit the probit model

```
probit_model=Probit(df.TenYearCHD,df[cols])
result=probit_model.fit()

print(result.summary())
print(result.pred_table())
for i in range(20):
   print(df.TenYearCHD.iloc[i],
result.predict(df[cols]).iloc[i])
```

Probit Regression

- Confusion matrix is now

```
[[3085.   16.]
 [ 547.   10.]]
```

- More false positives but less false negatives

Multinomial Logistic Regression

- Want to predict one of several categories based on feature vector
- Use the softmax function

$$\text{softmax}\left(z_1, z_2, \ldots, z_m\right) = \left(\frac{e^{z_1}}{\sum_{i=1}^{m} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{m} e^{z_i}}, \ldots, \frac{e^{z_m}}{\sum_{i=1}^{m} e^{z_i}} \right)$$

Multinomial Logistic Regression

- Learning is still possible, but more complicated

Multinomial Logistic Regression

```
model1 = LogisticRegression(random_state=0,
                            multi_class='multinomial',
                            penalty='none',
                            solver='newton-cg').fit(X_train, y_train)
preds = model1.predict(X_test)
```

Final Example

- Spine data

Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Class_att	
63.0278175	22.55258597	39.60911701	40.47523153	98.67291675	-0.254399986	0.744503464	12.5661	14.5386	15.30468	-28.658501	43.5123	Abnormal	
39.05695098	10.06099147	25.01537822	28.99595951	114.4054254	4.564258645	0.415185678	12.8874	17.5323	16.78486	-25.530607	16.1102	Abnormal	
68.83202098	22.21848205	50.09219357	46.61353893	105.9851355	-3.530317314	0.474889164	26.8343	17.4861	16.65897	-29.031888	19.2221	Abnormal	Prediction is done by using binary classification.
69.29700807	24.65287791	44.31123813	44.64413017	101.8684951	11.21152344	0.369345264	23.5603	12.7074	11.42447	-30.470246	18.8329	Abnormal	
49.71285934	9.652074879	28.317406	40.06078446	108.1687249	7.918500615	0.543360472	35.494	15.9546	8.87237	-16.378376	24.9171	Abnormal	
40.25019968	13.92190658	25.1249496	26.32829311	130.3278713	2.230651729	0.789992856	29.323	12.0036	10.40462	-1.512209	9.6548	Abnormal	Attribute1 = pelvic_incidence (numeric)

- Use explanations to give column names
- Remove last column

Final Example

```
back_data = pd.read_csv("spine.csv")
del back_data['Unnamed: 13']
back_data.columns = ['pelvic_incidence','pelvic tilt',
                     'lumbar_lordosis_angle','sacral_slope',
                     'pelvic_radius','degree_spondylolisthesis',
                     'pelvic_slope','Direct_tilt','thoracic_slope',
                     'cervical_tilt','sacrum_angle','scoliosis_slope',
                      'Status']
print(back_data.Status.describe())
```

Final Example

- Can also change the values of Status Column to 0 or 1

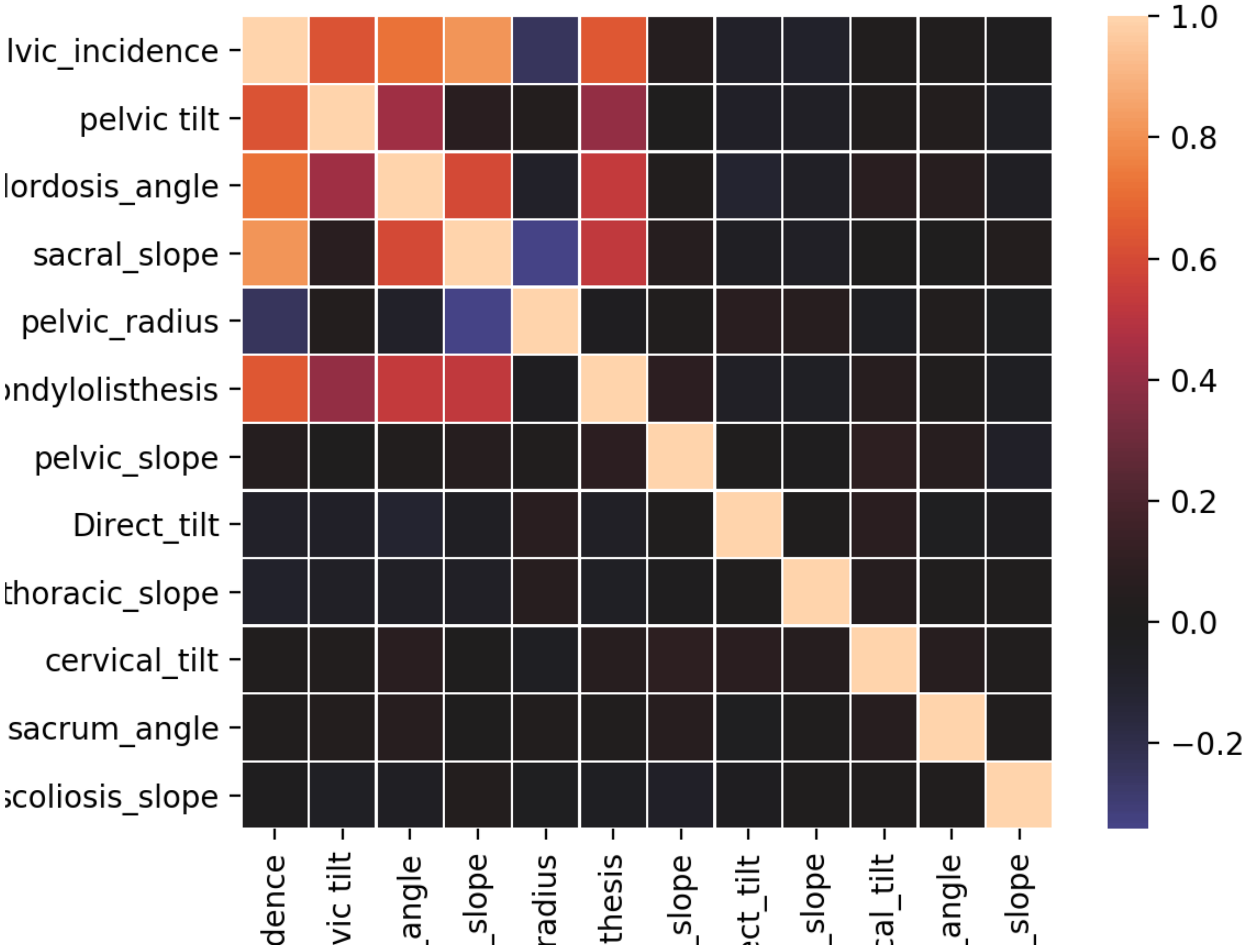
```
back_data.loc[back_data.Status=='Abnormal','Status'] = 1
back_data.loc[back_data.Status=='Normal','Status'] = 0
X = back_data.iloc[:, :12]
y = back_data.iloc[:, 12]
```

Final Example

- First task:
  - Are any of the columns strongly correlated?
    - Otherwise, model would have difficulties
    - Create a seaborn heatmap of the correlation

```
corr_back = back_data.corr()
sns.heatmap(corr_back, center=0, square=True,
linewidths=.5)
```

Final Example



Final Example

- We now see whether the values differ between normal and abnormal spines:

```
for x in back_data.columns[:-1]:
   print(x, back_data.groupby(‘Status').mean()[x])
for x in back_data.columns[:-1]:
   print(x, back_data.groupby(‘Status').median()[x])
```

Final Example

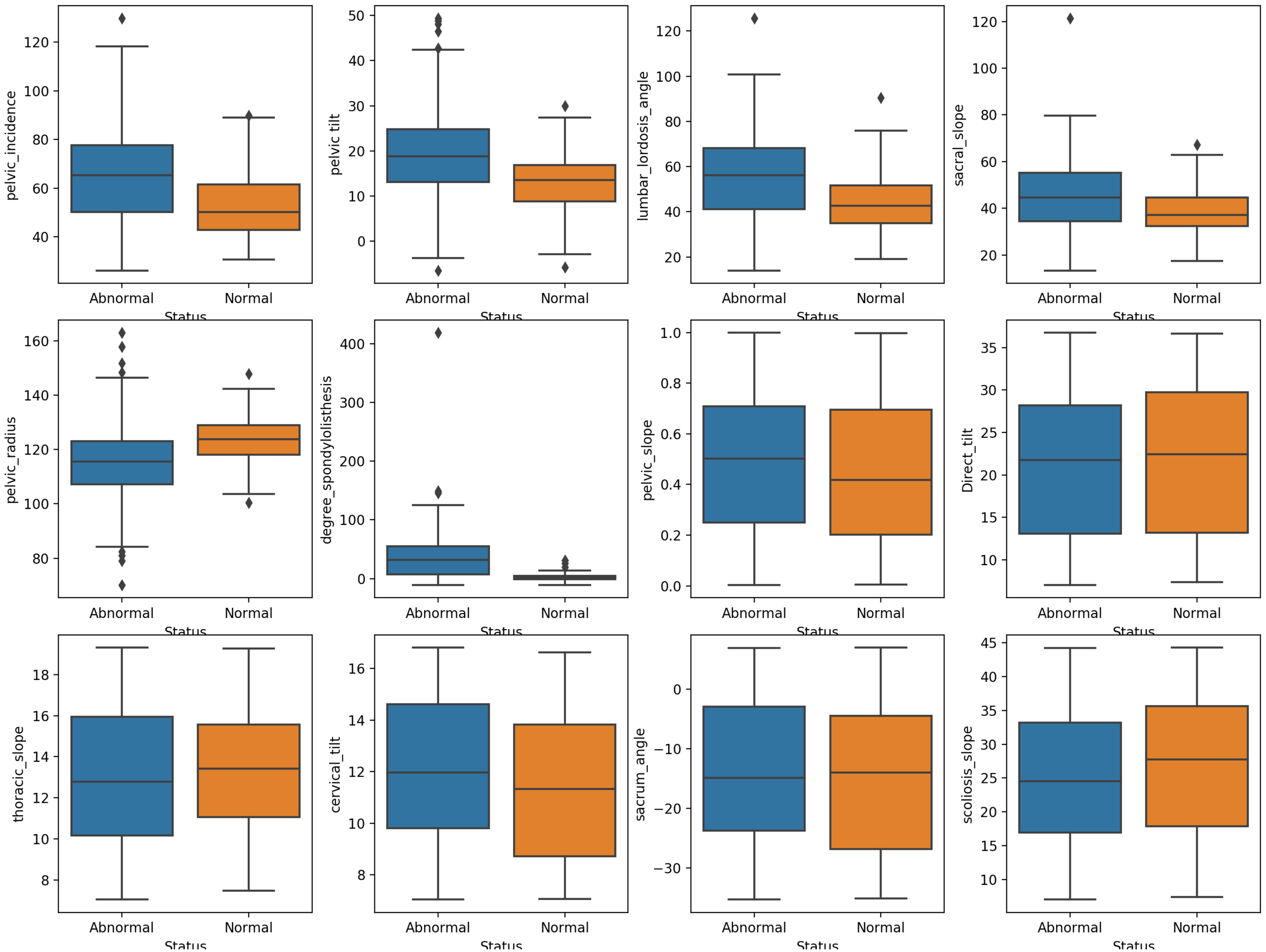
- Can also use a box plot to see the difference

```
fig, axes = plt.subplots(3, 4, figsize = (15,15))
axes = axes.flatten()

for i in range(0,len(back_data.columns)-1):
    sns.boxplot(x="Status", y=back_data.iloc[:,i],
                data=back_data, orient='v', ax=axes[i])

plt.tight_layout()
plt.show()
```

Final Example



Final Example

- Need to create training set and test set
- Need to scale:
  - Mean is set to 0
  - StDev is set to 1
  - Can be done with
    - sklearn.preprocessing.StandardScaler

Final Example

```
def data_preprocess(X,y):
    X_train, X_test, y_train,
        y_test = train_test_split(X, y.values.ravel(),
        test_size=0.3,
        random_state=0)
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler(copy=True,
                            with_mean=True,
                            with_std=True)

    scaler.fit(X_train)

    train_scaled = scaler.transform(X_train)
    test_scaled = scaler.transform(X_test)
    return(train_scaled, test_scaled, y_train, y_test)
```

Final Example

- We use the logistic regression model from sklearn

```
from sklearn.linear_model import LogisticRegression
```

-

```
X_train_scaled, X_test_scaled, y_train, y_test =
data_preprocess(X,y)
logreg = LogisticRegression().fit(X_train_scaled,
y_train)

logreg_result = logistic_regression(X_train_scaled,
y_train)
    logreg = LogisticRegression().fit(x, y)
```

Final Example

- We can now read the results:

```
logreg_result.score(X_train_scaled,y_train)
Training set score: 0.876
logreg_result.score(X_test_scaled,y_test)
Test set score: 0.817
```

Final Example

- To see what influence variables have, we use statsmodels

```
X_train_scaled, X_test_scaled, y_train, y_test =
data_preprocess(X,y)

logit_model = sm.Logit(y_train, X_train_scaled)
result = logit_model.fit()
print(result.summary2())
```

Final Example

```
                         Results: Logit
==================================================================
Model:              Logit            Pseudo R-squared: 0.248
Dependent Variable: y                AIC:              229.3058
Date:               2020-11-19 18:19 BIC:              269.8646
No. Observations:   217              Log-Likelihood:   -102.65
Df Model:           11               LL-Null:          -136.45
Df Residuals:       205              LLR p-value:      3.4943e-10
Converged:          0.0000           Scale:            1.0000
No. Iterations:     35.0000
------------------------------------------------------------------
     Coef.     Std.Err.      z    P>|z|      [0.025        0.975]
------------------------------------------------------------------
x1   0.0814 11580039.8359  0.0000 1.0000 -22696460.9366 22696461.0993
x2   0.0765  6600560.9760  0.0000 1.0000 -12936861.7142 12936861.8673
x3  -0.2797        0.3142 -0.8904 0.3733        -0.8955        0.3361
x4  -0.5412  9111339.5243 -0.0000 1.0000 -17857897.8597 17857896.7773
x5  -1.1234        0.2351 -4.7773 0.0000        -1.5842       -0.6625
x6   2.3250        0.4401  5.2832 0.0000         1.4625        3.1875
x7   0.1711        0.1790  0.9561 0.3390        -0.1797        0.5220
x8  -0.2115        0.1770 -1.1950 0.2321        -0.5583        0.1354
x9   0.0724        0.1738  0.4166 0.6770        -0.2683        0.4131
x10  0.2003        0.1772  1.1301 0.2584        -0.1471        0.5476
x11 -0.1042        0.1804 -0.5778 0.5634        -0.4578        0.2493
x12 -0.2749        0.1764 -1.5579 0.1193        -0.6207        0.0709
==================================================================
```

Final Example

- There was no convergence, meaning that there was some high correlation between variables
- Pelvic Incidence column is sum of Pelvic Tilt and Sacral Slope
- Let’s remove these

```
cols_to_include = [cols for cols in X.columns
                   if cols not in
      ['pelvic_incidence', 'pelvic tilt','sacral_slope']]
X = back_data[cols_to_include]
```

- And run again

Final Example

```
Optimization terminated successfully.
         Current function value: 0.481933
         Iterations 7
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.234
Dependent Variable: y                AIC:              227.1591
Date:               2020-11-19 18:23 BIC:              257.5781
No. Observations:   217              Log-Likelihood:   -104.58
Df Model:           8                LL-Null:          -136.45
Df Residuals:       208              LLR p-value:      8.5613e-11
Converged:          1.0000           Scale:            1.0000
No. Iterations:     7.0000
--------------------------------------------------------------------
       Coef.     Std.Err.       z       P>|z|      [0.025     0.975]
--------------------------------------------------------------------
x1    -0.5434     0.2568    -2.1158    0.0344    -1.0468    -0.0400
x2    -0.9642     0.2080    -4.6364    0.0000    -1.3719    -0.5566
x3     2.2963     0.4142     5.5443    0.0000     1.4846     3.1081
x4     0.1499     0.1771     0.8464    0.3974    -0.1972     0.4971
x5    -0.2442     0.1738    -1.4047    0.1601    -0.5849     0.0965
x6     0.0640     0.1732     0.3694    0.7118    -0.2754     0.4034
x7     0.2068     0.1747     1.1841    0.2364    -0.1355     0.5491
x8    -0.1183     0.1777    -0.6660    0.5054    -0.4666     0.2299
x9    -0.2872     0.1736    -1.6547    0.0980    -0.6274     0.0530
=================================================================
```

Final Example

- We concentrate on those variables with a low P-value:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
x1	-0.5434	0.2568	-2.1158	**0.0344**	-1.0468	-0.0400
x2	-0.9642	0.2080	-4.6364	**0.0000**	-1.3719	-0.5566
x3	2.2963	0.4142	5.5443	**0.0000**	1.4846	3.1081
x4	0.1499	0.1771	0.8464	0.3974	-0.1972	0.4971
x5	-0.2442	0.1738	-1.4047	0.1601	-0.5849	0.0965
x6	0.0640	0.1732	0.3694	0.7118	-0.2754	0.4034
x7	0.2068	0.1747	1.1841	0.2364	-0.1355	0.5491
x8	-0.1183	0.1777	-0.6660	0.5054	-0.4666	0.2299
x9	-0.2872	0.1736	-1.6547	0.0980	-0.6274	0.0530

Final Example

```
X_trim_1 = X.loc[:,['lumbar_lordosis_angle’,
                    'pelvic_radius',
                    'degree_spondylolisthesis']]

X_train_scaled, X_test_scaled, y_train, y_test =
data_preprocess(X_trim_1,y)
logit_model = sm.Logit(y_train, X_train_scaled)
result = logit_model.fit()
print(result.summary2())
```

Final Example

```
=====================================
Optimization terminated successfully.
         Current function value: 0.498420
         Iterations 7
                         Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.207
Dependent Variable: y                AIC:              222.3145
Date:               2020-11-19 18:30 BIC:              232.4542
No. Observations:   217              Log-Likelihood:   -108.16
Df Model:           2                LL-Null:          -136.45
Df Residuals:       214              LLR p-value:      5.1622e-13
Converged:          1.0000           Scale:            1.0000
No. Iterations:     7.0000
-------------------------------------------------------------------
       Coef.    Std.Err.      z       P>|z|     [0.025    0.975]
-------------------------------------------------------------------
x1    -0.4688     0.2426   -1.9325   0.0533   -0.9443    0.0067
x2    -0.9188     0.2037   -4.5100   0.0000   -1.3181   -0.5195
x3     2.1897     0.3937    5.5626   0.0000    1.4182    2.9613
=================================================================
```

Final Example

- Some improvement in scores:
-
```
Training set score: 0.857
Test set score: 0.774
```