Machine Learning

Lecture 05

Logistic Regression

Logistic Regression

- Logistic Regression learning
  - It is a linear and (essentially) a binary classification model (even though it’s called “regression” – yes, a misnomer!)
  - An extension of linear regression to classification problems with a class probability estimate
  - It estimates the probability of occurrence of an event by fitting data to a "logit" function and hence the name "logit regression" as well
  - Question:
    - Can we model the probability of a target class as a linear function of a predictor?

Characteristics of LR

- Learning method: Supervised Learning (parametric and discriminative)
- Types of LR: Binomial & Multinomial LR
- Handling of mixed data types: Yes
- Requires linearly separable data? No
- Classification ability: Binary (can be extended to Multi-class via Multinomial LR)
- Computationally demanding? No
- Ability for online classification: Yes
- Interpretability: High
- When to consider: Good for clinical trials, scoring and fraud detection (where response is binary), measuring the effectiveness and success rates of marketing campaigns, predicting the customer churn.

Logistic Regression formulation

- **Example**: Probability of credit approval as a function of income

  Target variable : **Y** (credit approval) = $\begin{cases} 1 & \text{(credit approved)} \\ 0 & \text{(credit denied)} \end{cases}$

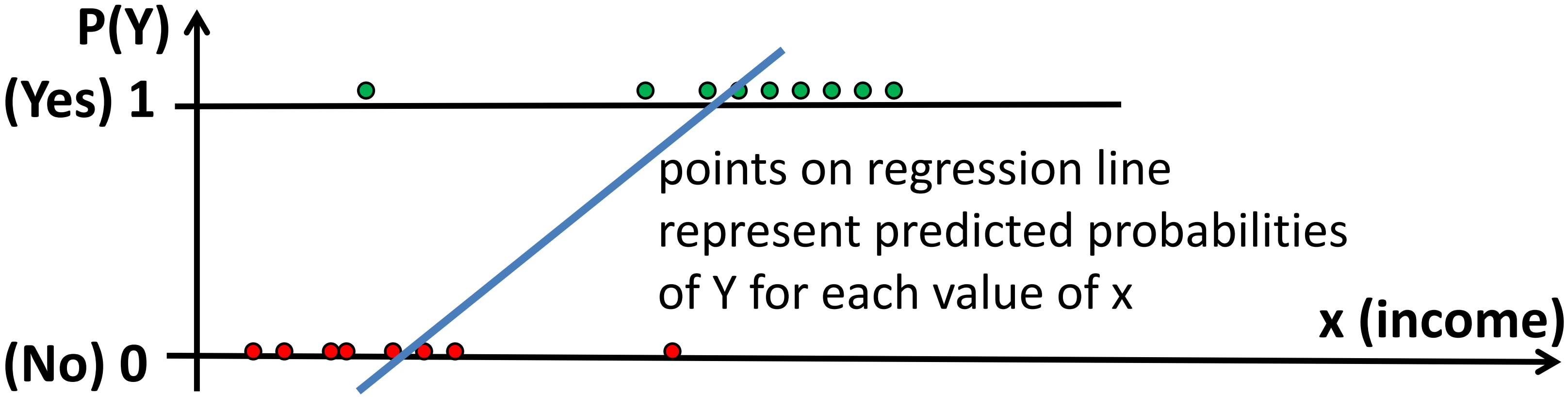
  Predictor : **x** (income)

  – Given the income x, the probability that the credit is approved is expressed by a linear model:

  $$\boxed{P(Y=1\,|\,x)=\beta_0+\beta_1 x}$$

  Positive event (Y=1) refers to the event we want to predict

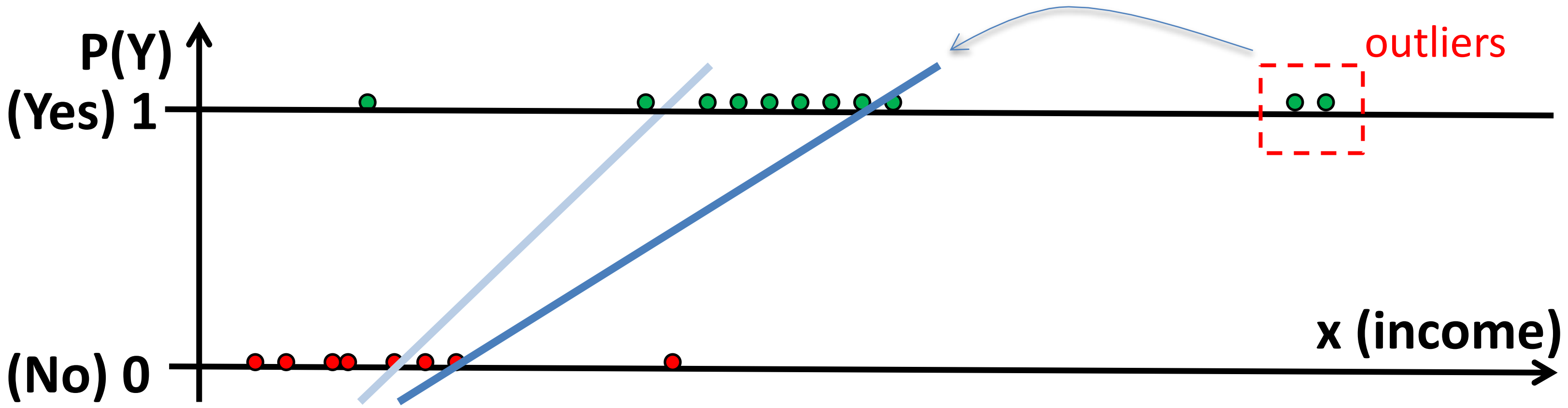
  – How about using a least squares fit?



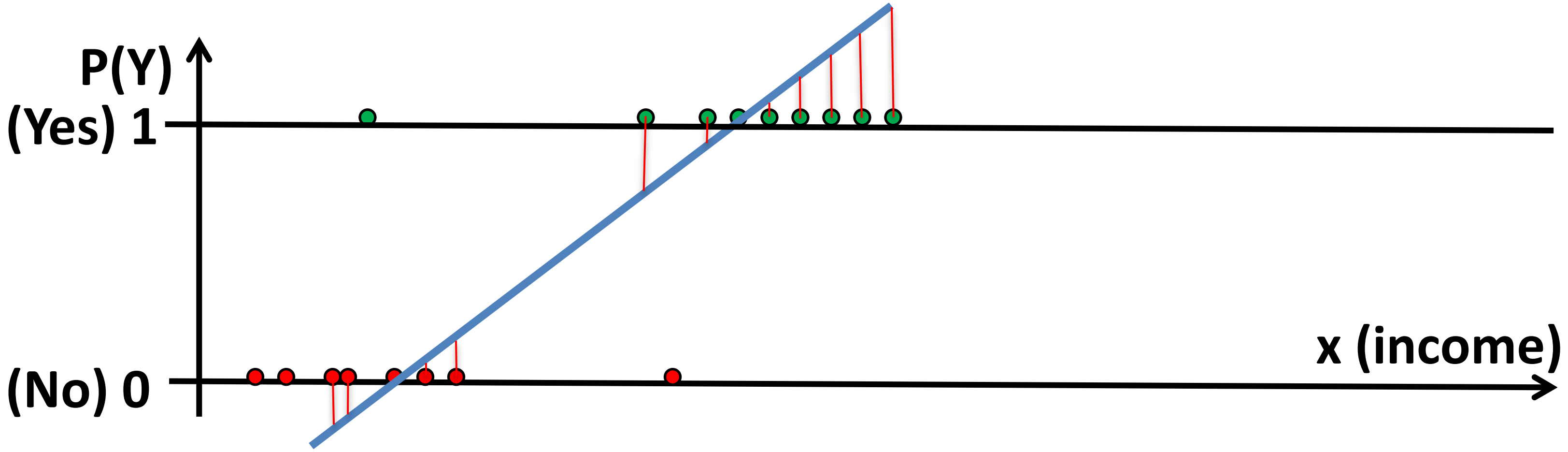
points on regression line represent predicted probabilities of Y for each value of x

LoR formulation – cont'd

- What is wrong with this approach (if anything)?
  - Distant points dominated the fit causing misclassifications.



  - A clear pattern in the error terms around the regression line



LoR formulation – cont'd

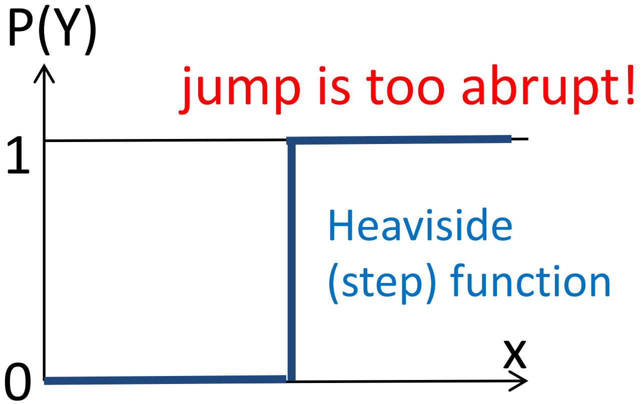
- What is wrong with this approach (if anything)?

  Right hand side of the linear equation (**β_0 + β_1x**) can take any value

- Probability, however, must be between [0,1]

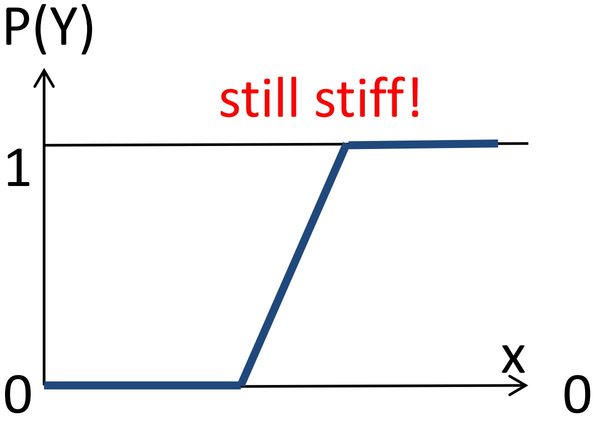
$$\boxed{P(Y=1\,|\,x)=\beta_0+\beta_1 x} \in (-\infty,+\infty)$$

- So, this linear function **p = f(input) = β_0+β_1x** didn't work!
- Can we improve the model to fix these problems?



jump is too abrupt!

Heaviside (step) function



still stiff!

LoR formulation – cont’d

- We’ll go through a 2-step process:

  1. It must always be positive (as p ≥ 0), so how about $p = e^{\beta_{10} + \beta_1 x}$ which is always positive?
  2. It must also be ≤1, so what if we do:

$$\boxed{p = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}}$$

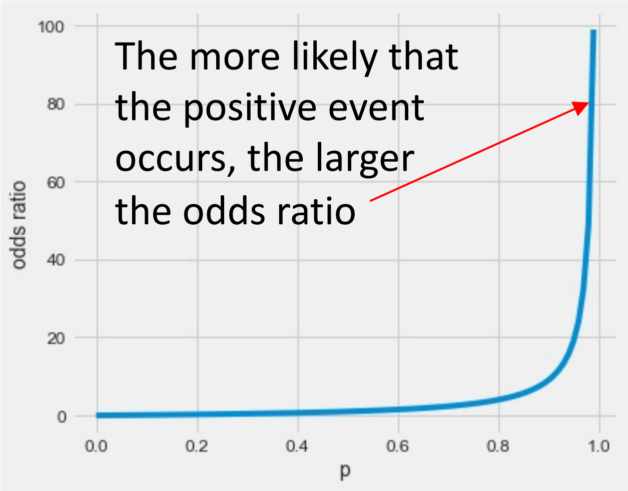
which yields a p such that 0 ≤ p ≤ 1

- Although this expression looks quite complex, the linearity has not completely gone. How?
- Let's introduce the concept of "**odds**":
- It's the ratio of chances of success (something happening) to chances of failure (something not happening)

$$odds = \frac{\text{probability of event occurring}}{\text{probability of event NOT occurring}} = \frac{P(Y=1)}{P(Y=0)} = \frac{p}{1-p}$$

LoR formulation – cont'd

- Suppose: wins 🟢🟢🟢🟢 losses 🔴🔴🔴🔴🔴🔴
- Probability of winning = 4/10 (4 out of 10)
- Odds of winning = 4/6 (4 to 6)
- Assuming I'm a bad player, I keep playing 100 times and still win only 4 games. My odds of winning is 4/96=0.042. The worse I play, my odds of winning moves towards zero while the odds of losing extends to infinity.



The more likely that the positive event occurs, the larger the odds ratio

This asymmetry is quite unappealing, because the odds of winning should be the opposite of the odds of losing.

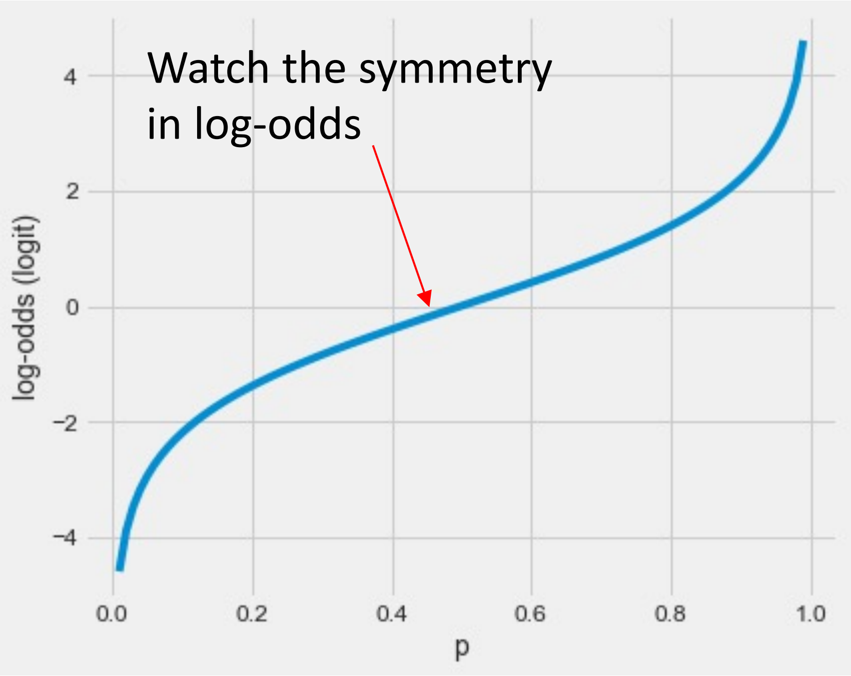
How can we fix this?

LoR formulation – cont'd

- By taking the natural log of both sides of this odds ratio we get the log-odds (aka **logit function**):

$$\mathbf{odds} = \frac{p}{1-p}$$

$$\boxed{\ln(\mathbf{odds}) = \ln(\frac{p}{1-p}) = \ln(e^{\beta_0+\beta_1 x}) = \beta_0 + \beta_1 x}$$



Watch the symmetry in log-odds

Although the probability of credit approval is not a linear function of x, the logit is:

$$\boxed{\ln(\frac{p}{1-p}) = \beta_0 + \beta_1 x}$$

The linearity of the logit helps us to apply the standard regression vocabulary:
"If X is increased by 1 unit, the *logit* of Y changes by 1"

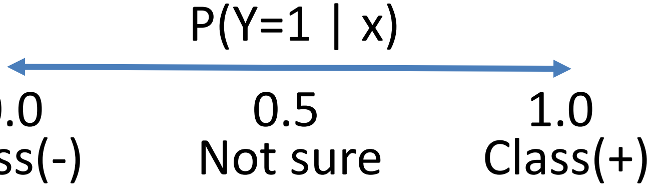
LoR formulation – cont’d

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x$$

But we're not interested in the right hand side of the above equation as we want to find p(Y=1|x)

- So by taking the inverse of this logit function, we get the logistic sigmoid (a nonlinear relationship between the probability and predictors) which returns the class probabilities:

$$\boxed{P(Y=1|x) = p = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}}$$

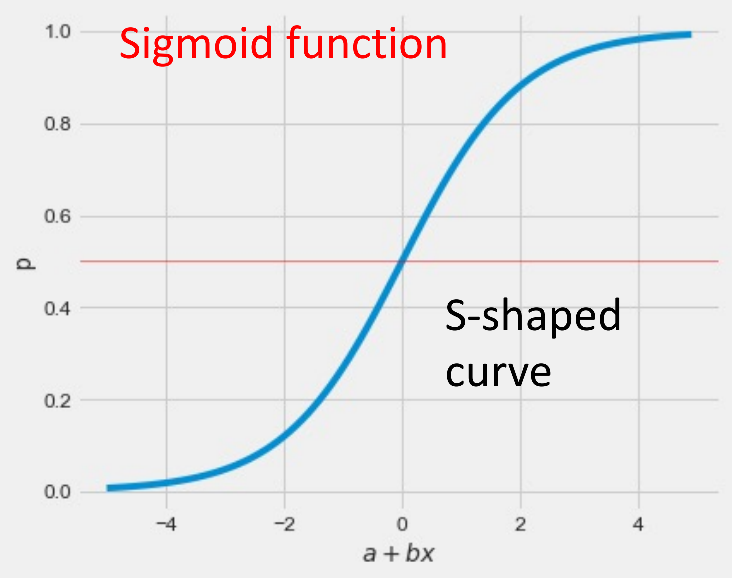


P(Y=1 | x)

0.0
Class(-)

0.5
Not sure

1.0
Class(+)



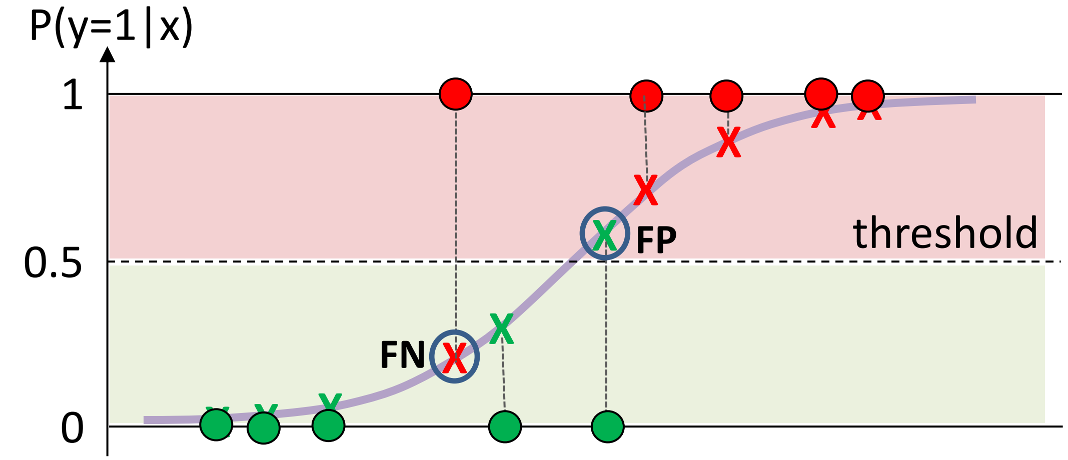
Sigmoid function

S-shaped curve

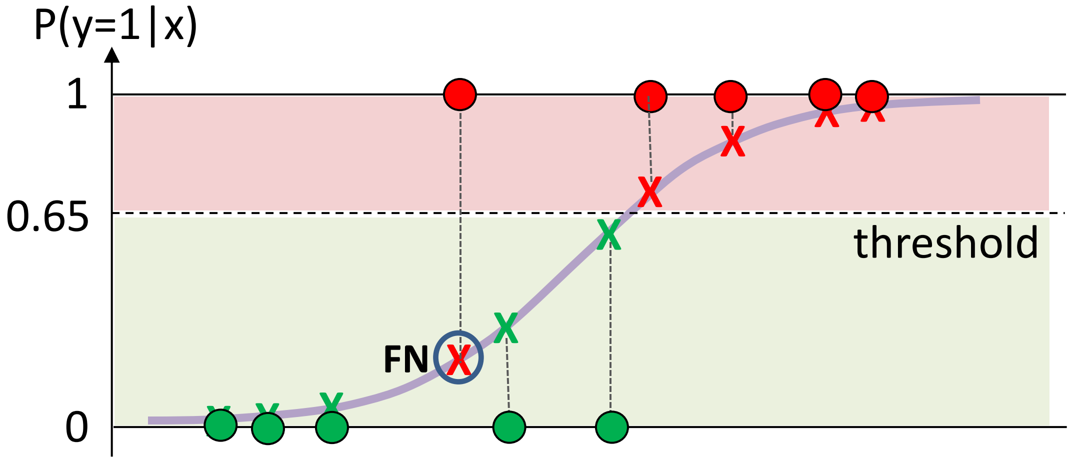
Asymptotically getting closer to zero

How to interpret Logistic Regression output

y => 1: Heart Attack (HA), 0: No HA
x => Cholesterol level (mg/dl)



Accuracy : 8/10 = 80%
False Positive : 1
False Negative: 1



Accuracy : 9/10 = 90%
False Positive : 0
False Negative: 1

Multiple Logistic Regression

- Profiling for advertising: Shopping preference in terms of consumer demographics
- Data

Gender	Married	Income	Age	Preference
Female	No	$ 30K	21	Online shopping
Male	No	$ 75K	30	Offline shopping
…	…	…	…	…
Female	Yes	$ 120K	41	Offline shopping

two classes

- Categorical variables are transformed into numerical values for regression analysis:

Gender: Male = 0
Gender: Female = 1

Married: No = 0
Married: Yes = 1

- Regression analysis in the form of:

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 Gender + \beta_2 Married + \beta_3 Income + \beta_4 Age$$

Multiple LoR – cont'd

- How do we find the coefficients β_i?
- In its most general form:

$$\ln(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_N x_N = \sum_{i=0}^{N} \beta_i x_i$$

- In vector notation:

$$\ln(\frac{p}{1-p}) = \beta^T \mathbf{x} \quad \text{where} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ . \\ \beta_N \end{pmatrix} \quad \text{and} \quad \mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ . \\ . \\ . \\ x_N \end{pmatrix}$$

- So, the hypothesis is written as:

$$\boxed{P(Y=1 \mid x) = p = \frac{1}{1+e^{-\beta^T \mathbf{x}}}}$$

Calculation of coefficients

- How do we compute $\boldsymbol{\beta}_j$?
- OLS uses the minimum least squares method to estimate coefficients. In LoR, however, the relation between Y and x is nonlinear. So, the least squares is inappropriate for LoR.
  - The dependent variable Y (or the residuals) doesn't follow a normal distribution (Y is a categorical response variable and it takes only the values of 0 and 1, following a binomial distribution)
  - The variance of Y is not constant across all classes (high variance for p near 0.5 and lower variance towards 0 and 1)
- LR therefore uses the **Maximum Likelihood Estimation** method to estimate coefficients that maximize the likelihood of the training data.

Calculation of coefficients (cont'd)

- MLE method and the cost function?
- Construct a likelihood function that expresses the probability of observed events as a function of α and β

$$P(Y=1|x)=\frac{1}{1+e^{-\beta^T\mathbf{x}}}=F_\beta(x)$$

$$P(Y=0|x)=1-F_\beta(x)$$

- Likelihood for one observation (in a more compact form):

$$P(Y=y|x)=[F_\beta(x)]^y[1-F_\beta(x)]^{1-y}$$

which is a Binomial distribution where y is either 0 or 1.

- As the observations are assumed to be independent, the likelihood function for the whole dataset is the multiplication of the individual data points:

Calculation of coefficients (cont'd)

- Likelihood (objective) function for N observations from a population: $L(\beta) = \prod_{i=1}^{N} [F_\beta(x_i)]^{y_i} [1 - F_\beta(x_i)]^{1-y_i}$
- Turn products into sums (easier to deal with) by taking the "log" of both sides and come up with the (cross entropy) cost (hence the minus sign) function:

$$\log(L) = J(\beta) = -\sum_{i=1}^{N} \left[y_i \log(F_\beta{}^{(i)}) + (1 - y_i) \log(1 - F_\beta{}^{(i)}) \right]$$

- We actually turned the objective function (which we would try to maximize) into a cost function (which we are trying to minimize) by converting it into the negative log likelihood function: To fit the coefficients, we will **iteratively** find β_j's that minimize the cost function:

$$\arg\min_\beta \left[J(\beta) \right]$$ ➔ Get β_j

Calculation of coefficients (cont'd)

- Optimization methods:
  - Gradient Descent
  - Stochastic Gradient Descent
  - Newton Method
  - and others
- **Gradient Descent:**

$$\boxed{\frac{\partial J}{\partial \beta_j} = \sum_{i=1}^{N} \left(F_\beta(x_i) - y_i \right) [x_i]_j}$$

repeat {

$$\beta_j := \beta_j - \delta \frac{\partial J}{\partial \beta_j}$$

}

(by simultaneously updating all θ_j)

$\boxed{\delta \text{ : learning rate}}$

- To make a prediction given a new x^*:

$$P(Y = 1 | x^*) = \frac{1}{1 + e^{-\beta^T \mathbf{x}^*}}$$

Assumptions of LR

- Unlike Linear Regression, **it doesn’t need a linear relationship between the DV and the IV’s**. It can handle all sorts of relationships as it applies a nonlinear log transformation to the predicted odds ratio.
- **IV’s do not need to be multivariate normal** (though it can yield more stable solutions)
- **The residuals don’t need to be normally distributed**
- **Homoscedasticity is not needed**. LR doesn’t need variances to be heteroscedastic for each level of IV’s
- Can handle ordinal and nominal data as IV’s
- IV’s don’t need to be metric (interval or ratio)

Assumptions of LR (cont’d)

- Dependent variable should be binary (dichotomous)
- The model should be fitted correctly (no under- or over-fitting), i.e., all relevant features should be included.
- The error terms need to be independent
- It’s required that **each observation be independent** and the model has little or **no multicollinearity**
- The logit transformation of the outcome variable has a linear relationship with the predictor variables. No influential observations (Outliers)
- Requires large sample sizes **-** (at least 10-15 observations per feature)

Assumptions of LR (cont’d)

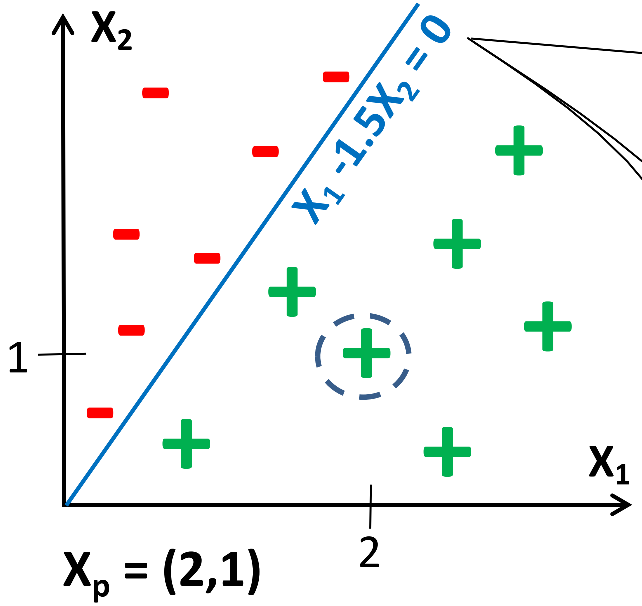
- What is the effect of collinearity on Logistic Regression?
- The effect is quite similar to the case in Linear Regression:

1. Parameter estimates (coefficients) have large variances
2. The model becomes unstable. Small variations in data can cause large variations inparameter estimates, reversing their signs, making an insignificant attribute significant when the collinear feature is removed etc.

- If the objective is just a prediction, i.e., classifying the observation, then collinearity may not be a problem. You'll lose, however, much of the interpretability in your model because of the highly variant parameter estimates.

LoR with linearly separable data

- When the data is linearly separable, MLE estimation drives coefficients (β) to larger and larger values to emphasize the difference between the two classes:



Data linearly separable
with coeff's (1) and (-1.5)

$$p(Y=1\,|\,X_p) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2)}}$$

$$= \frac{1}{1+e^{-(0+1*2-1.5*1)}} = 0.62$$

Data linearly separable
with coeff's (10) and (-15)

$$p(Y=1\,|\,X_p) = 0.99$$

Data linearly separable
with coeff's (10^8) and ($-1.5\text{x}10^8$)

$$p(Y=1\,|\,X_p) \approx 1$$

LoR and Overfitting

- As we see in the previous slide, if data is linearly separable, weights go to infinity leading to non-convergence.
- Solution? Penalizing weights to keep them from growing
- We had the likelihood function earlier:

$$L = \prod_{i=1}^{N} P(Y=1 \mid x_i)^{y_i} [1 - P(Y=1 \mid x_i)]^{1-y_i}$$

- We'll now solve the optimization problem subject to an added constraint λ (regularization parameter):

$$L + \lambda \sum_i \beta_i^2$$ L2 regularization (Ridge)

$$L + \lambda \sum_i |\beta_i|$$ L1 regularization (Lasso)

- Adding a regularization term will have the effect of simplifying the models and improve the performance in presence of overfitting/multicollinearity.

LoR and Overfitting – cont'd

- When regularization parameter

  $\lambda \to \infty$: All coefficients shrink to zero

  $\lambda \to 0$: Unregularized Logistic Regression

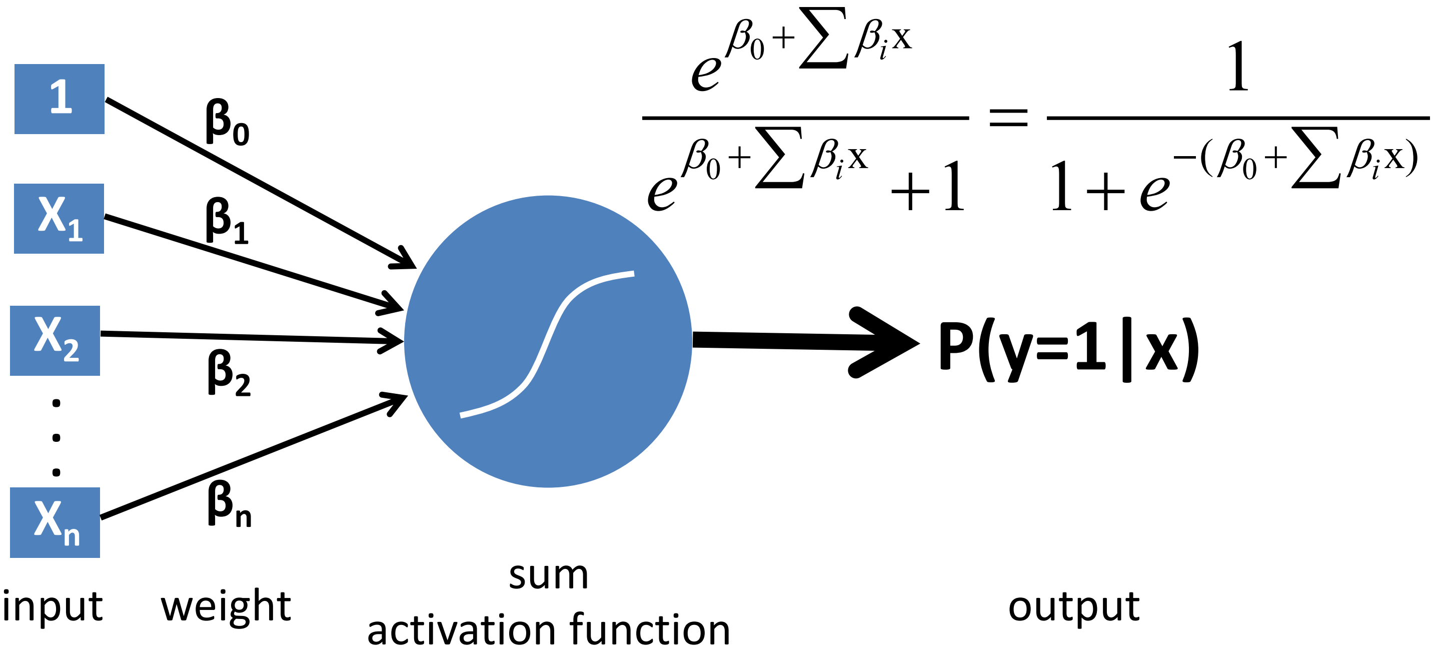
- How to find the optimum λ?
  - A cross validation over a range of values for λ will reveal the optimum value of λ where validation error is a minimum.
  - Scikit implementation of regularization:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(C=0.1, penalty='l2')
LR.fit(X_train_scaled, y_train)
```

  - In scikit-learn λ is implemented as $C = 1/ \lambda$. So a value of 0.1 for C is $\lambda=10$.
  - If you don't need regularization, assign a very large value to C such as 10^6.

Why is Logistic Regression a linear model?

- Schematic of a Logistic Regression classifier

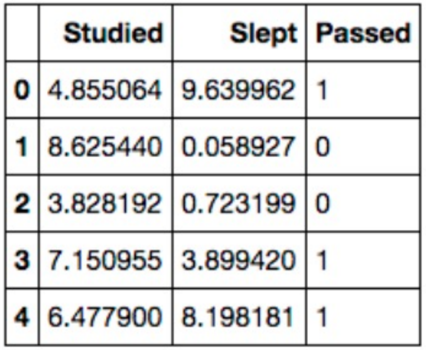


$$\frac{e^{\beta_0+\sum\beta_i x}}{e^{\beta_0+\sum\beta_i x}+1}=\frac{1}{1+e^{-(\beta_0+\sum\beta_i x)}}$$

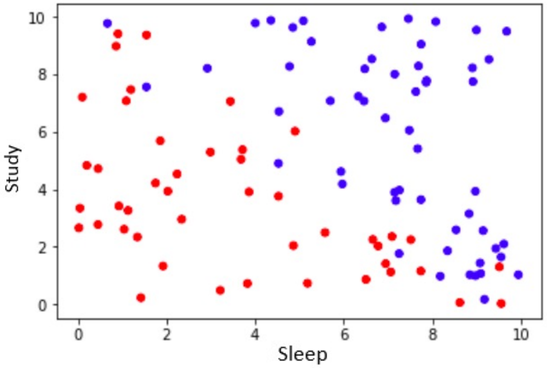
- The logistic regression classifier has a non-linear activation function, but the outcome always depends on a linear combination (sum) of the inputs and parameters (coefficients), which is why logistic regression is a "**generalized**" linear model.

Why is Logistic Regression a linear model?

- **Example**: Assume we're predicting exam performance given the time spent on studying and sleeping:



	Studied	Slept	Passed
0	4.855064	9.639962	1
1	8.625440	0.058927	0
2	3.828192	0.723199	0
3	7.150955	3.899420	1
4	6.477900	8.198181	1



$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \text{Study} + \beta_2 \text{Sleep}$$

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \text{Study} + \beta_2 \text{Sleep})}}$$

- LogisticRegression() from scikit-learn yields:

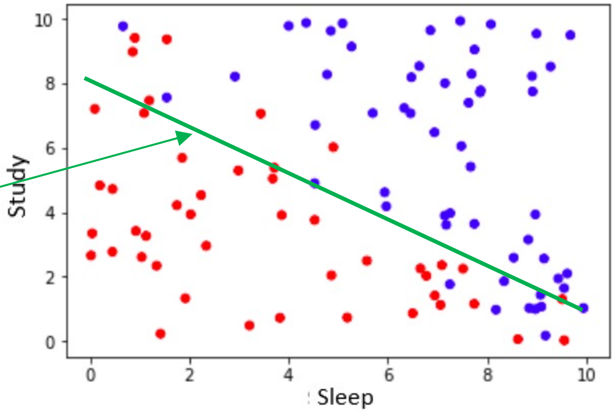
```
Intercept: -3.767
Coefficients: (0.474, 0.338)
```

- Decision boundary is when P(Y=1|x) = 0.5 (decision threshold):

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \text{Study} + \beta_2 \text{Sleep}$$

```
log(0.5/(1-0.5)) = 0
= -3.767+0.474*Study+0.338*Sleep
```

```
Study = 8 – 0.7*Sleep
```



Ref: towardsdatascience.com/whats-linear-about-logistic-regression-7c879eb806ad

More on Logistic Regression

- Can LoR have nonlinear decision boundaries?
  - Using higher order terms and interactions for features can allow LoR to learn more complex (nonlinear) decision boundaries. But overfitting becomes a risk for such configurations.
- Do we need to scale the features?
  - Scaling for Logistic Regression, however, brings all features to the same scale increasing the convergence of the numerical approximation (Gradient Descent solution) and improves the overall stability of the solution.
  - When you use regularization (L1 or L2), Hastie, Tibshirani and Friedman (An Introduction to Statistical Learning) in their book point out that one should scale (normalize/standardize) features before the solution (page 63).
  - Logistic Regression is sensitive to outliers. So scaling the features would also reduce the impact of outliers. So if in doubt, scale your data, it doesn't hurt.

Solver types in LR

- Scikit-learn uses the following solvers:
- '**newton-cg**' : Slow for large data sets
- '**lbfgs**' : Not super fast for large data sets, but very efficient in memory allocated (default in sklearn 0.22+)
- '**liblinear**' : Used to be the default solver for its robustness. Good performance on high dimensional data. No parallelization. When used, multi-class solution is available with 'ovr' only.
- '**sag**' : Fast for large data sets
- '**saga**' : Extended 'sag' that allows for L1 regularization. Usually faster than 'sag'. Both 'sag' and 'saga' require that the features be on a similar scale (scaling needed)

Other types of LR

- **Simple Logistic Regression**
  - Logistic Regression is a binary classifier in nature
  - For simple Logistic Regression, the dependent variable is dichotomous (binary) – as we covered so far.
- **Ordinal Logistic Regression**
  - Dependent variable has more than 2 classes
  - These classes follow an order, i.e.:
    - [Hot, warm, cool, cold], [Small, medium, large]
- **Multinomial Logistic Regression**
  - Dependent variable has more than 2 classes
  - No specific order for classes (like ordinal regression with the order being ignored)
    - Red, Blue, Green

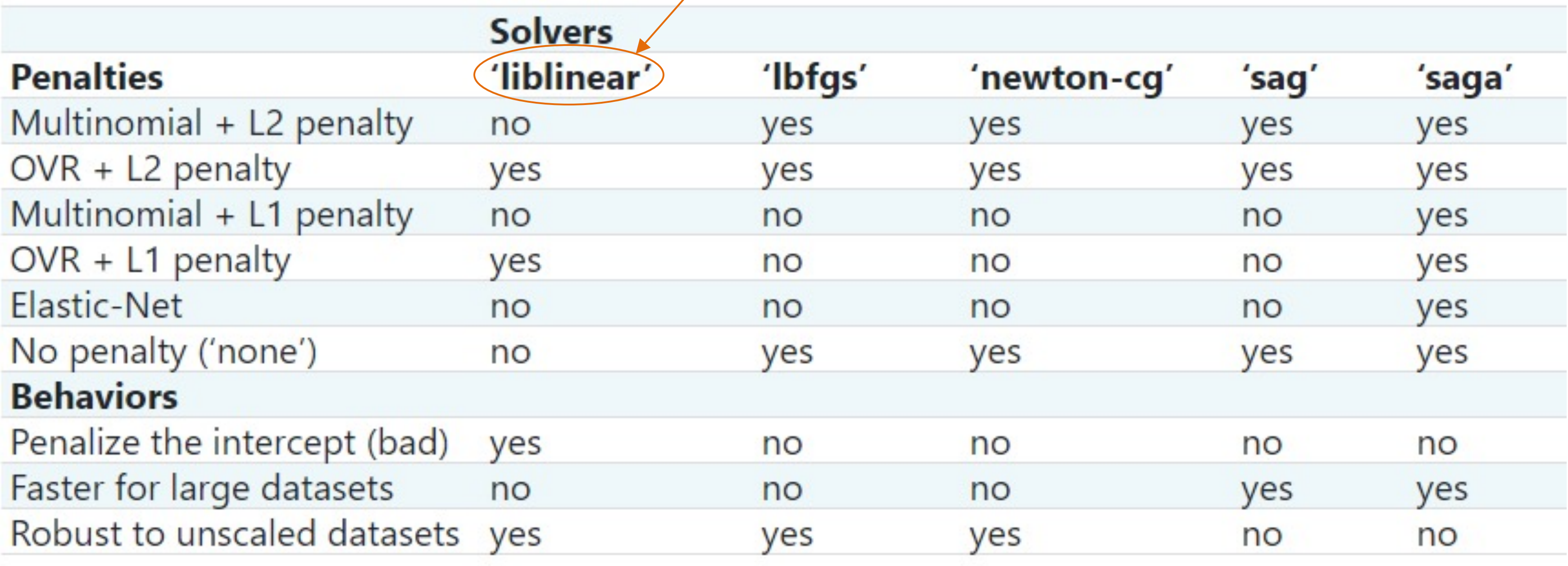
Other types of LR

- **How does Logistic Regression handle multi-class?**
- Scikit-learn has the following parameter: **multi_class**
- For multi class problems, you can set the **multi_class** to **multinomial**. In this case, multinomial loss (fitted across the entire probability space) is minimized.
  - multinomial is unavailable when solver = '**liblinear**'
- If **multi_class** = '**auto**', '**auto**' selects '**multinomial**' unless:
  - The class is binary, or the solver is '**liblinear**' , in which case the selection is '**ovr**' (one-versus-rest: a binary problem is fit for each label)
  - For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss;
  - 'liblinear' is limited to one-versus-rest schemes.

Penalty vs solver

- We don't have every combination of penalty parameter (L1 and L2) and solver pair available in the scikit-learn library. Table below summarizes the relation between the two:

A good choice for small datasets



	Solvers				
Penalties	**‘liblinear’**	**‘lbfgs’**	**‘newton-cg’**	**‘sag’**	**‘saga’**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty (‘none’)	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Ref: https://scikit-learn.org/stable/modules/linear_model.html

Stochastic Gradient Descent

- Stochastic Gradient Descent (SGD) considers only 1 random point while changing weights unlike gradient descent which considers the whole training data. As such stochastic gradient descent is **much faster** than gradient descent when dealing with large data sets.
- What is a SGDClassifier? SGD is indeed an optimization method. So a SGDClassifier is a linear classifier which implements regularized linear models optimized by the SGD.
- For example:
  - An implementation of Logistic Regression

    ```
    SGDClassifier(alpha=k, penalty='l2', loss='log')
    ```

  - An implementation of Linear SVM (later)

    ```
    SGDClassifier(alpha=k, penalty='l2', loss='hinge')
    ```

  where alpha is the regularization term (default = 0.0001)

Pros and Cons of LR

- **Pros**
  - Easy to implement and interpret
  - Doesn’t require linear relationship between DV and IV’s
  - Classification with probabilistic estimates (ranking)
  - No assumptions about distributions of classes in feature space
  - Fast to train and apply, computationally inexpensive & efficient
  - Performs well with a few attributes
  - Tells you which attributes are more important
  - Can adjust the classification thresholds

- **Cons**
  - Generally, requires large sample sizes as the MLE is less powerful at low sample sizes than ordinary least squares
  - With small training data, model estimates may overfit the data
  - May have low accuracy
  - Suffers from multicollinearity (independent variables should not be correlated with each other)
  - Sensitive to outliers
  - Linear decision boundary (poor perf. on linearly inseparable data)

Logistic Regression w/ Python

```python
# using scikit-learn library
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# c: optimum regularization parameter found via CV
Clf = LogisticRegression(penalty='l2', C=c)
Clf.fit(X_train, y_train)
accuracy_score(y_test, Clf.predict(X_test))

# Hyperparameters tuned
# C: Regularization strength
# penalty: L1 or L2 (Lasso or Ridge regularization)
# Default: C=1.0, penalty=L1
# Can handle multiple classes out-of-the-box
```

Calibration of Probabilities

Probabilities

- Class probabilities are a useful part of machine learning models. Here is the probability estimate from a Logistic Regression model: $P(Y=1|x) = p = \dfrac{1}{1+e^{-\left(\beta_0+\sum_i \beta_i x_i\right)}}$
- Why are probabilities important?
  - High risk cases: compare
    - Model predicted you don't have cancer
    - Model predicted you're 48% likely to have cancer
  - When a probability threshold is used for class assignment
  - Critical misclassifications due to
    - False labels with high probabilities
    - True labels with low probabilities
- So we may need not only predicted classes, but also accurate predicted probabilities

Probabilities – cont'd

- Scikit-learn provides the following for probability estimates:

```
Clf.fit(X_train, y_train)
p = predict_proba(X_train)
```

- But how good are these probability estimates?
- Are they model dependent?
- What properties of data degrade these estimates?
- But first of all, how do we know if our predicted probabilities are any good?
  - We'll use the Calibration curve (aka reliability diagram) for assessing the reliability of our predicted probabilities.
- What do we do if the predicted probabilities are not reliable?

Probabilities – cont'd

- To obtain meaningful predicted probabilities, we need to conduct what is called **calibration**.
- The distribution of the probabilities can be adjusted to better match the expected distribution observed in the data. This adjustment is referred to as **calibration**.
- **Calibration** is post-processing a model to improve its probability estimates
- A model is perfectly calibrated if, for any p, a prediction of a class with confidence p is correct 100p% of the time
- But remember: Accuracy ≠ Calibration
  - You may have a well-calibrated model with low accuracy (random coin flips)
  - You may have a high accuracy model with bad calibration

Probabilities – cont'd

- **Example of a reliability diagram:**

Predicted probability

True label

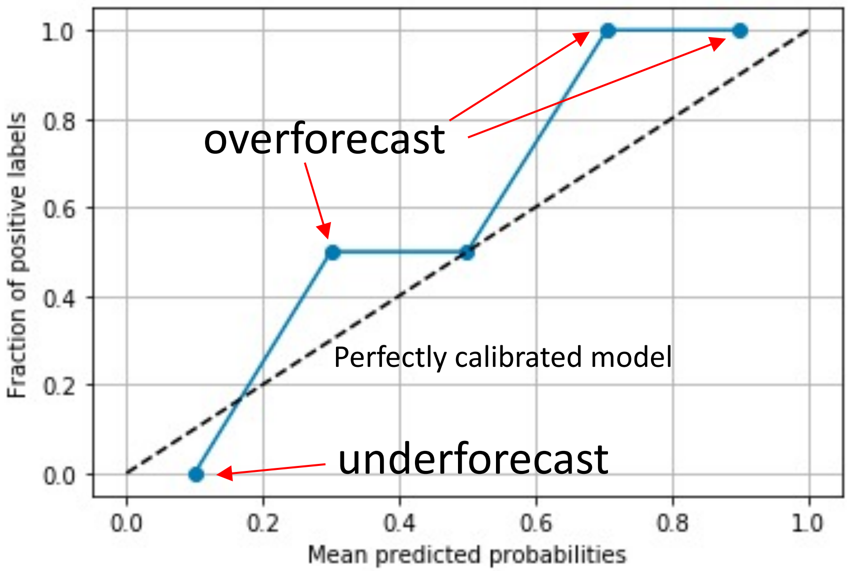
p̂(y)	y
0.85	1
0.65	1
0.35	1
0.75	1
0.55	1
0.15	0
0.25	0
0.95	1
0.05	0
0.45	0

=> sort

p̂(y)	y	
0.95	1	bin #5
0.85	1	
0.75	1	bin #4
0.65	1	
0.55	1	bin #3
0.45	0	
0.35	1	bin #2
0.25	0	
0.15	0	bin #1
0.05	0	

Predicted probability = (0.95+0.85)/2 = 0.90
Fraction of positive labels: 2 out of 2 (1.0)

For bin size of 5



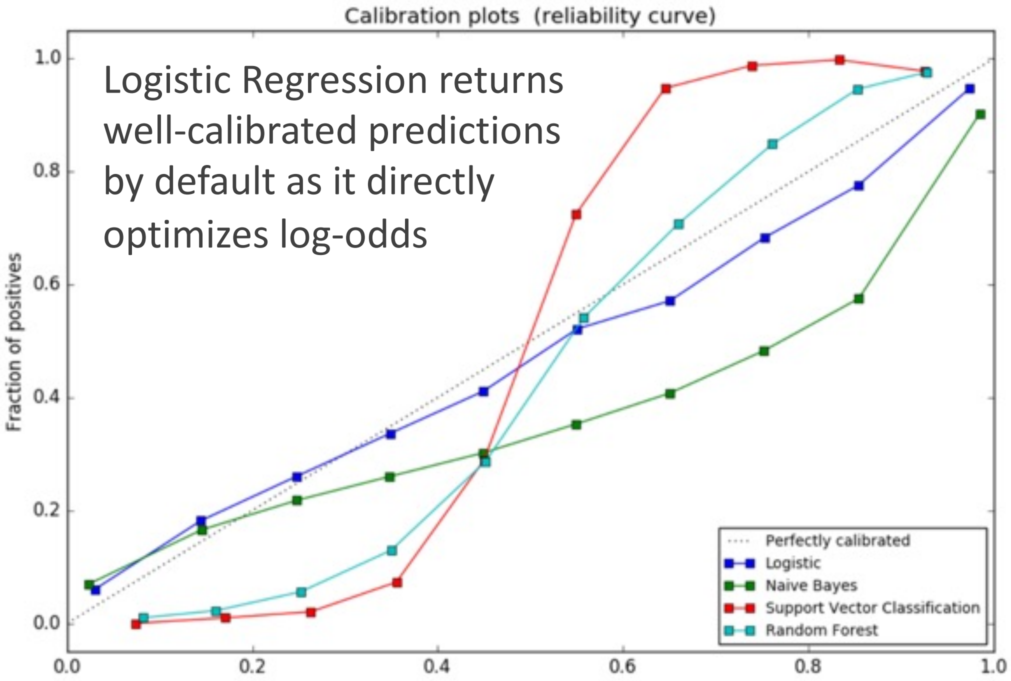
```
y_test = [0,0,0,1,0,1,1,1,1,1]
probs = [0.05,0.15,0.25,0.35,0.45,0.55,0.66,0.75,0.85,0.95]
prob_true, prob_pred = calibration_curve(y_test, probs, n_bins=5)
plt.plot(prob_pred, prob_true, '-o'); ident=[0,1]
plt.plot(ident, ident, linestyle='dashed', color='k')
```

Probabilities – cont'd

- Inaccurate probabilities can result in for different reasons:
  - Flawed model assumptions
    - e.g., independence assumption in Naive-Bayes (redundant / correlated features violate independence assumption)
  - Features that are unavailable at the time of training
  - Nature of the learning algorithm

SVM, Naive-Bayes, and Boosted Trees usually produce inaccurate probability predictions.

Logistic Regression, Random Forest and Neural Networks, however, tend to have well-calibrated probabilities.



Logistic Regression returns well-calibrated predictions by default as it directly optimizes log-odds

Probabilities – cont'd

- Do we care about well-calibrated probabilities?
  - If the objective is to rank observations from most likely to least likely, we don't need calibration.
  - if the required output is the true probability returned from a classifier whose probability distribution does not match the expected distribution of the predicted class
- Not all classifiers produce well-calibrated probabilities and, for some classifiers, the predicted probability does not match the output of its decision function.
- Logistics Regression is known to produce well-calibrated probabilities if and only if the parametric assumptions of the model holds. Naive Bayes is notorious for producing probabilities close to its extremes i.e. 0 and 1, whilst boosting and bagging techniques are known for producing probabilities away from 0 and 1, hence requiring calibration.

Methods of Calibration

- **Platt's Scaling**
  - Platt scaling involves fitting a Logistic Regression model to the probability output of the classifier with respect to true class labels.
  - Assumes each class probability is normally distributed
  - Platt scaling should not change the rank of the observations, so measures like accuracy and AUC remain unchanged (but log-loss will improve).
  - **Pros**: Works well with a small dataset
  - **Cons**: Could produce even worse probabilities if the assumptions do not hold.
  - Most effective when the distortion in the predicted probabilities is sigmoid-shaped.

Methods of Calibration – cont'd

- **Isotonic Regression**
  - This is a non-parametric approach which tends to perform well as it fits a piece wise non-decreasing function to the predicted probabilities outputted by the model.
  - Isotonic Regression is a more powerful calibration method that can correct any monotonic distortion.
  - **Pros:** Makes no assumption about the input probabilities.
  - **Cons**: Requires more data points to work well. A learning curve analysis shows that Isotonic Regression is more prone to overfitting, and thus performs worse than Platt Scaling, when data is scarce.