Computer Science

CSC380: Principles of Data Science

Basic machine learning 2

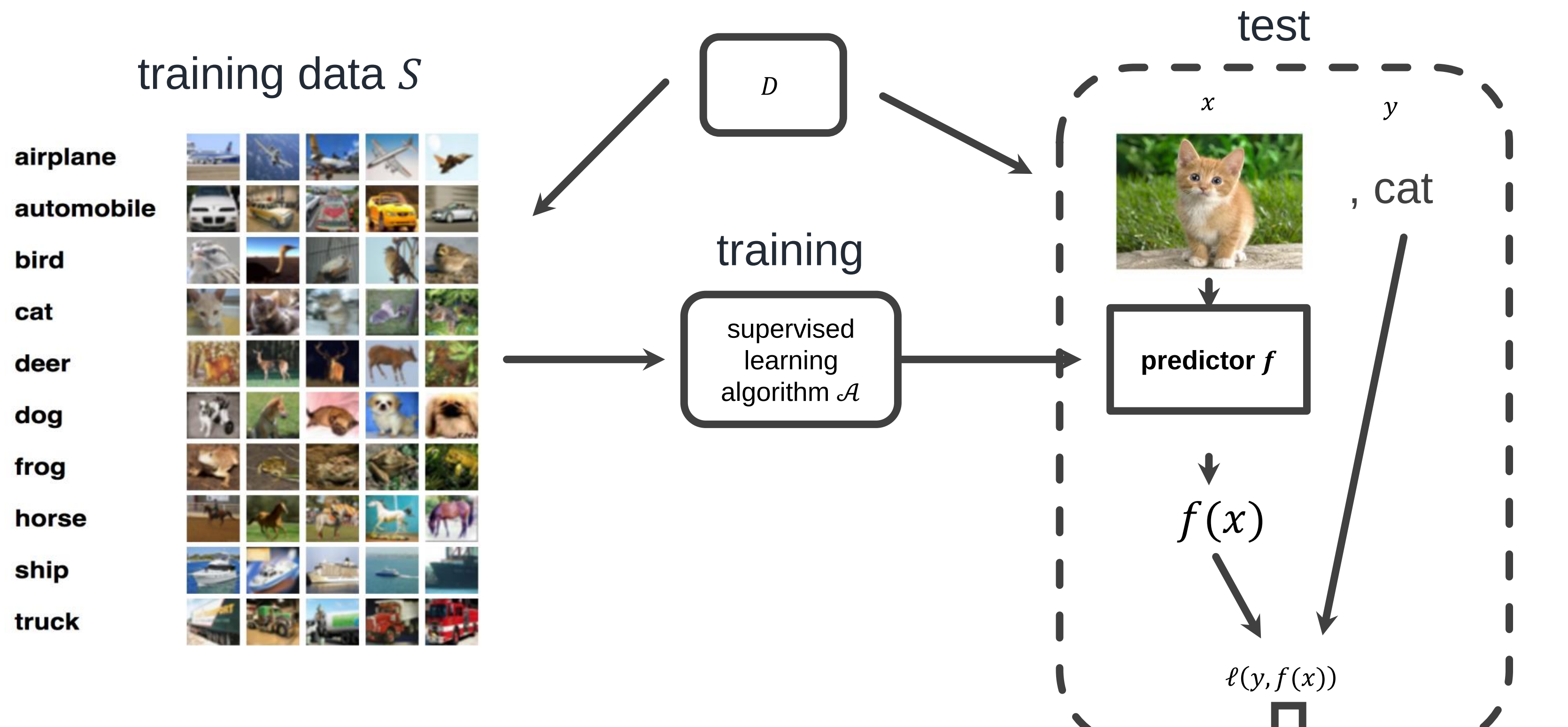
Chicheng Zhang

Outline

- Classification basics
- Nearest neighbor Classification
- Logistic regression

Classification recap

Supervised learning setup in one figure



- Goal: design learning algorithm $\mathcal{A}$, such that:
- after training, its output predictor f has low test error

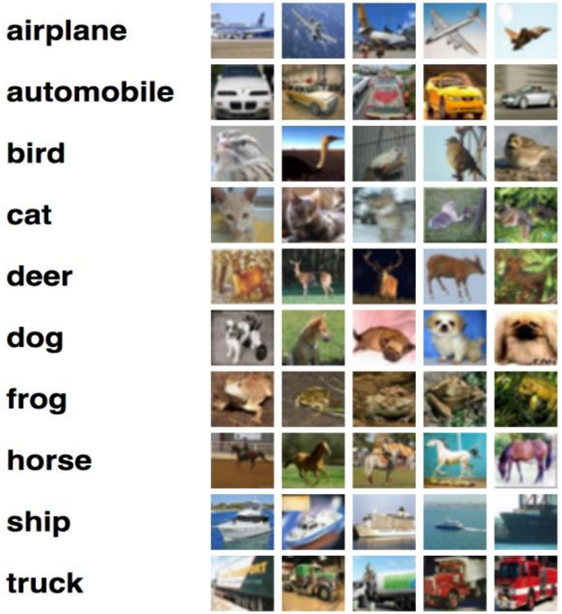
Test error: average of $\ell(y, f(x))$ in test set

Classification

- The labels are categorical

- Loss function ℓ: measures the quality of prediction $\hat{y}$ respect to true label y

- $\ell(y, \hat{y}) = I(y \neq \hat{y})$
- I: indicator of predicate; 1 if true; 0 if false



Nearest Neighbor Classification

Example: Course Recommendation

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Label: "like" (rows +2 to 0)

Label: "dislike" (rows -1 to -2)

Features (columns Easy? to Morning?)

Suppose we'd like to build a recommendation system for classes

We've collected information about many past classes

We can frame this as a classification problem:

Predict like/dislike from class features

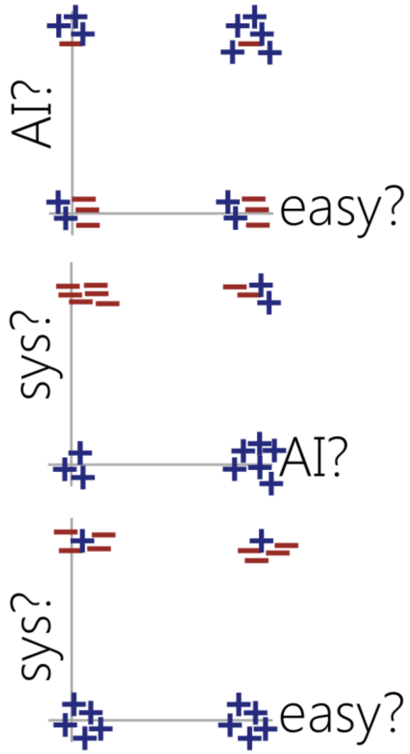
Example: Course Recommendation

Label: +

Label: -

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Features



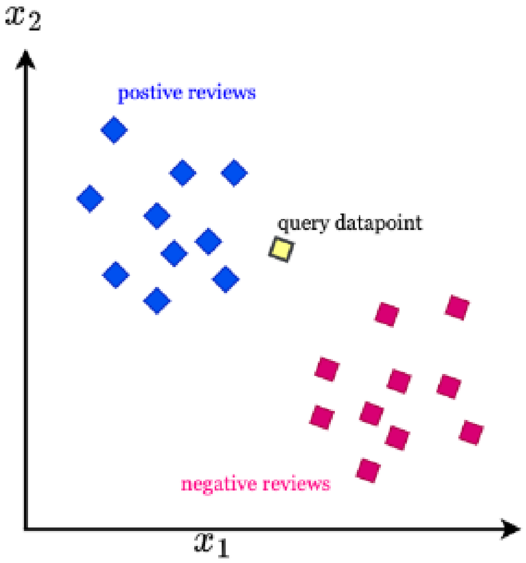
Each course's feature is Represented as points in 5-dimensional space

That's too many dimensions to plot…so we look at 2D projections…

Observation: examples with same labels tend to be closer!

Nearest neighbor classification

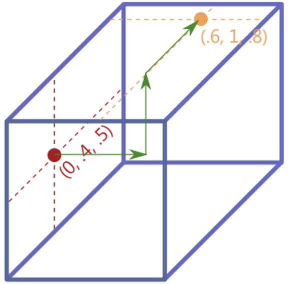
- Given a new course, would like to predict its label (+/-)
- Idea: Find its most similar course in the training set, and use that course's label to predict



Measuring nearest neighbors

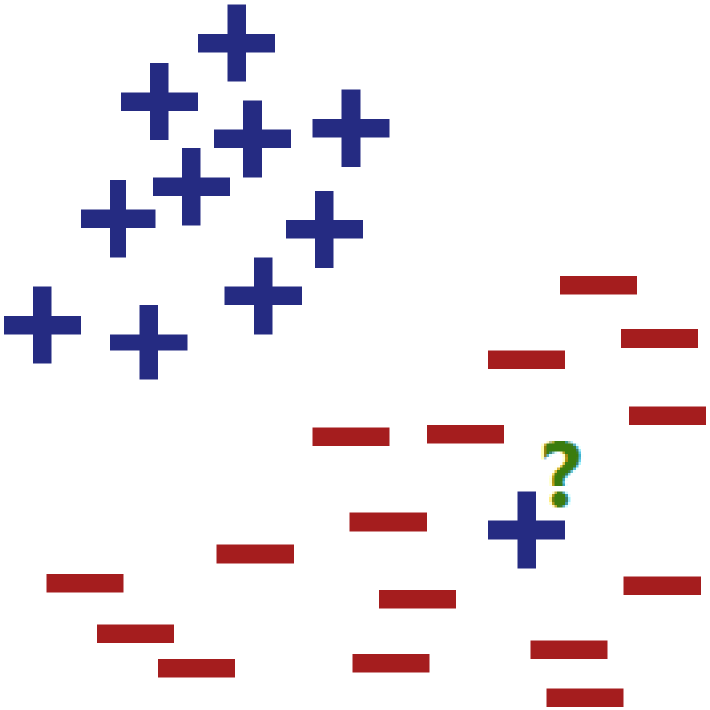
- Oftentimes convenient to work with feature $x \in \mathrm{R}^d$
- Distances in R^d: notation $x(f)$: $x = (x(1), \ldots, x(d))$
  - (popular) Euclidean distance $d_2(x, x') = \sqrt{\sum_{f=1}^{d} \left(x(f) - x'(f)\right)^2}$
  - Manhattan distance $d_1(x, x') = \sum_{f=1}^{d} |x(f) - x'(f)|$

  - If we shift a feature, would the distance change?
  - What about scaling a feature?



- How to extract features as **real values**?
  - Boolean features: {Y, N} -> {0,1}
  - Categorical features: {Red, Blue, Green, Black}
    - Convert to {1, 2, 3, 4}?
    - Better one-hot encoding: (1,0,0,0), .., (0,0,0,1) (IsRed?/isGreen?/isBlue?/IsBlack?)

Robustify Nearest Neighbor Classification



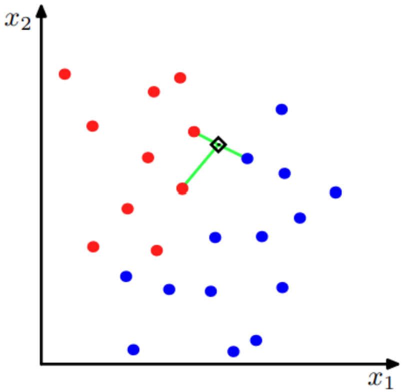
Query point ? Will be classified as + but should be -

Problem: predicting using 1 nearest neighbor's label can be sensitive to noisy data

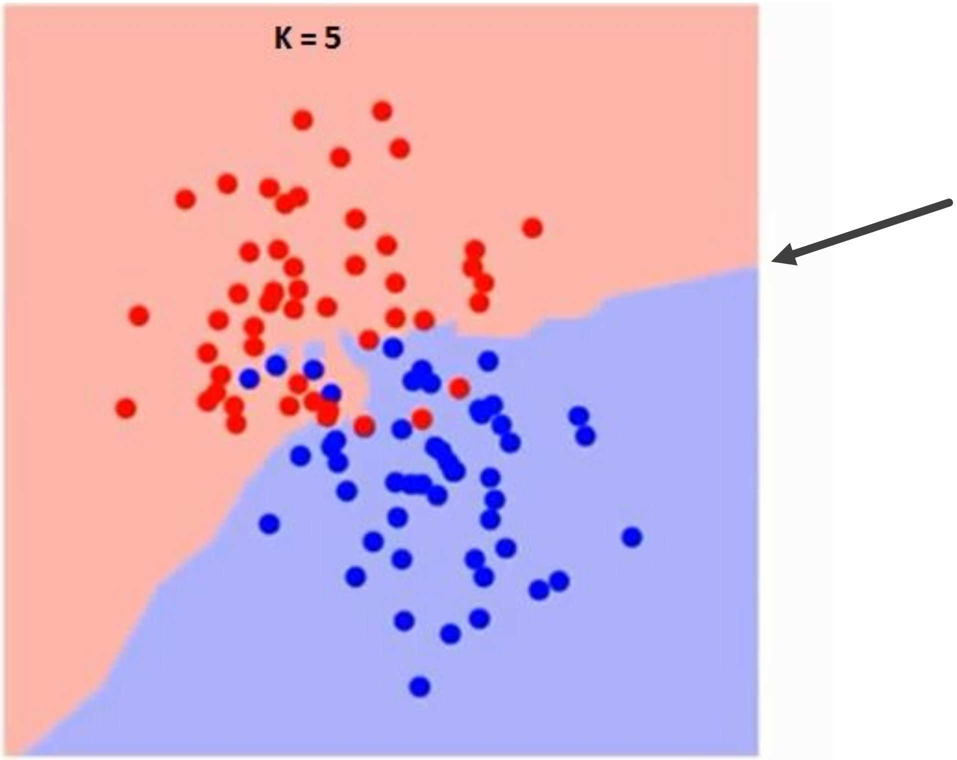
How to mitigate this?

k-nearest neighbors (k-NN): main concept

- Training set: $S = \{ (x_1, y_1), \ldots, (x_m, y_m) \}$
- **Key insight**: given test example x, its label should resemble the labels of *nearby points*
- Function
  - input: x
  - find the k nearest points to x from S; call their indices $N(x)$
  - output:
    - (classification) the majority vote of $\{y_i : i \in N(x)\}$
    - (regression) the average of $\{y_i : i \in N(x)\}$



k-NN classification example

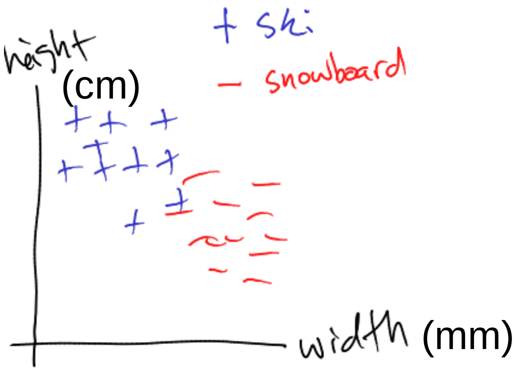


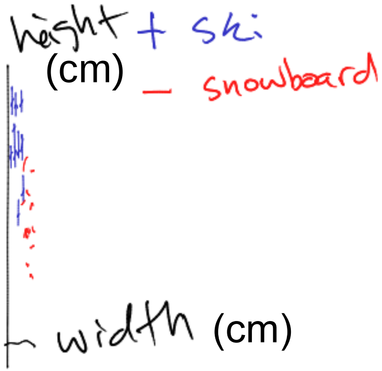
decision boundary

Issue 1: scaling

- Features having different scales can be problematic.

- Ex: ski vs. snowboard classification





- One solution: feature standardization

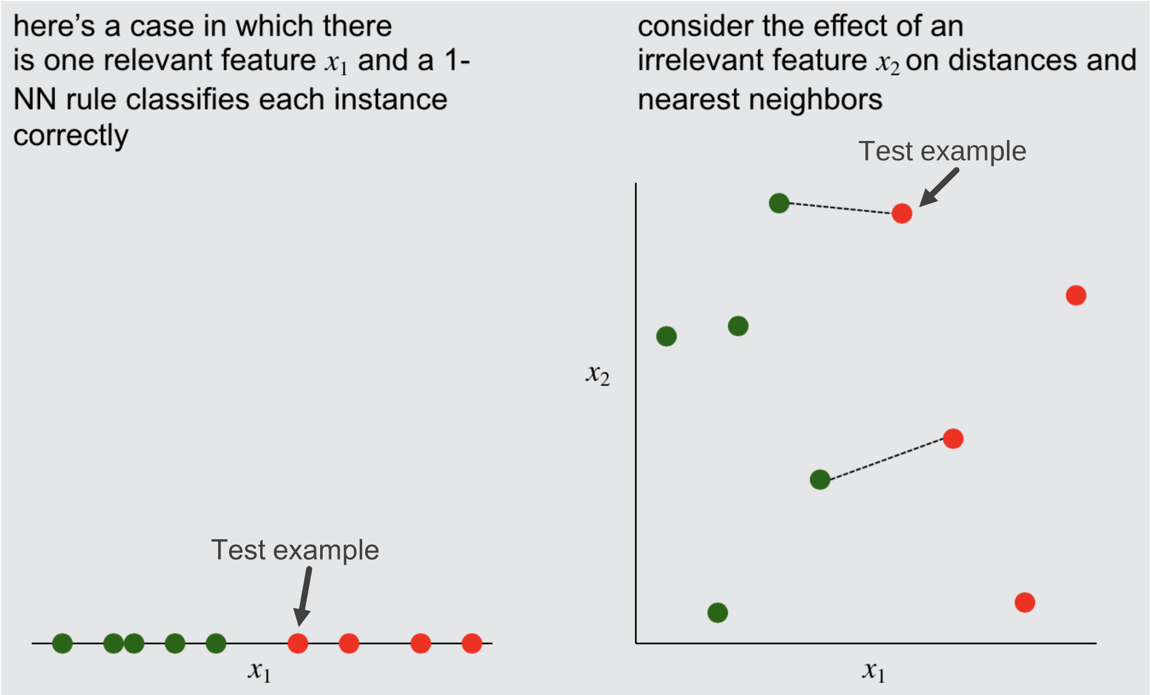
Make sure features are scaled fairly

- Features having different scale can be problematic
- [Definition] **Standardization**
  - For each feature f, compute $\mu_f = \frac{1}{m}\sum_{i=1}^{m} x_f^{(i)}$, $\sigma_f = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_f^{(i)} - \mu_f\right)^2}$
  - Then, transform the data by $\forall f \in \{1,\dots,d\}, \forall i \in \{1,\dots,m\}$, $x_f^{(i)} \leftarrow \frac{x_f^{(i)} - \mu_f}{\sigma_f}$

after transformation, each feature has mean 0 and variance 1

- Be sure to keep the “standardize” function and apply it to the test points.
  - Save $\{(\mu_f, \sigma_f)\}_{f=1}^{d}$
  - For test point x^*, apply $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$

Issue 2: irrelevant features



- Mitigation: feature selection

Issue 3: choosing k

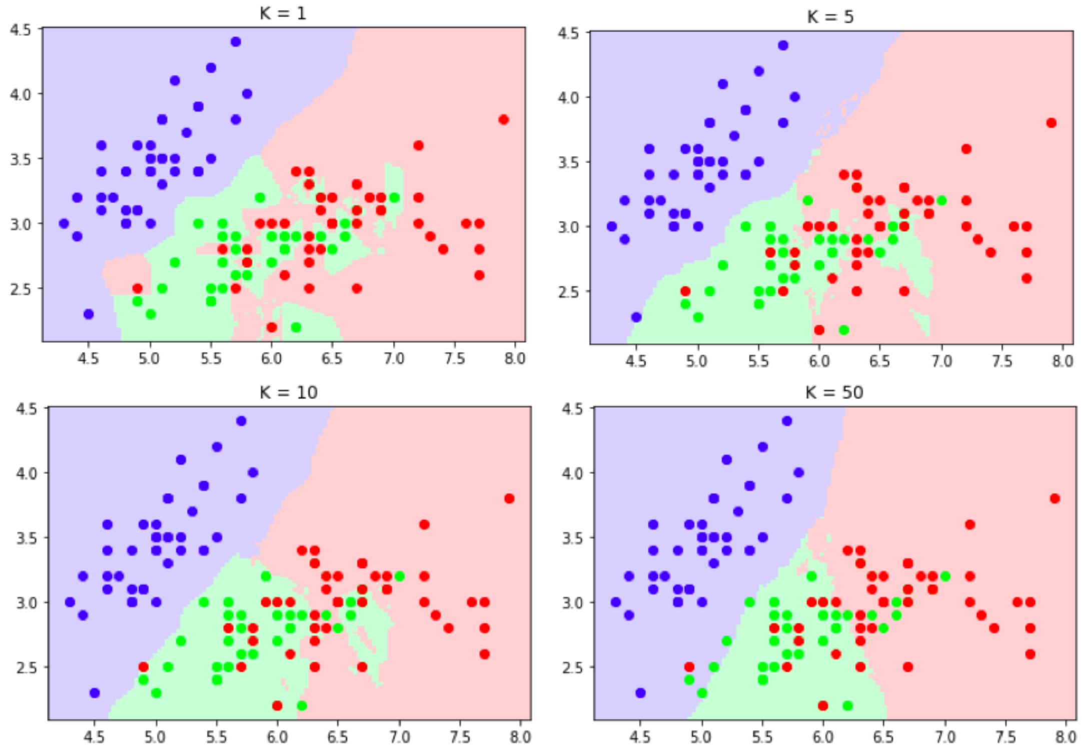
- Q: If we set $k = m$, then what classification rule does it look like?
  - Predict majority label everywhere
  - Underfitting

- Q: If we set $k = 1$, what would be the training error (assume there is no repeated train data point)?
  - 0
  - Overfitting

Issue 3: choosing k

k can be viewed as a model complexity measure

Smaller k results in a more complex model

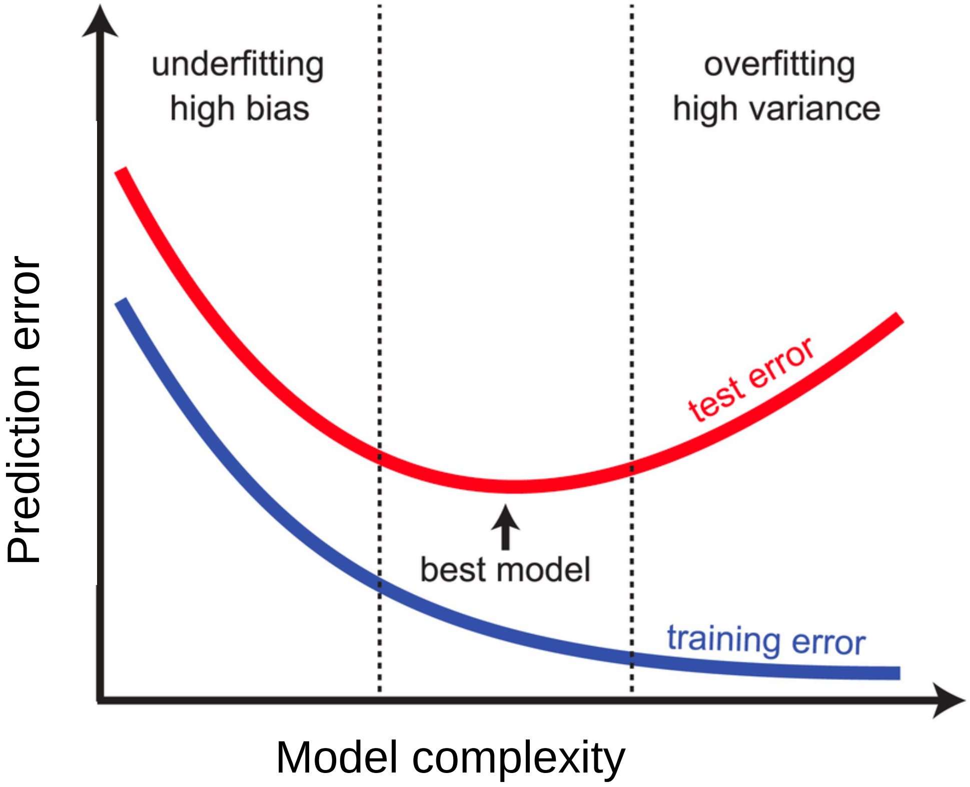


Issue 3: choosing k

We'd like to choose appropriate k to balance model bias and complexity

We can choose k in the same way we chose λ in ridge regression

- "Default" approach: cross validation



Scikit-learn nearest neighbors

```
class sklearn.neighbors.NearestNeighbors(*, n_neighbors=5, radius=1.0,
algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None,
n_jobs=None)
```
[source]

Unsupervised learner for implementing neighbor searches.

```
# 1. Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels (species)

# 2. Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Create the KNN classifier model
knn = KNeighborsClassifier(n_neighbors=3)  # Use 3 nearest neighbors

# 4. Train the model on the training data
knn.fit(X_train, y_train)
```

Scikit-learn nearest neighbors

```
# 5. Make predictions on the test set
y_pred = knn.predict(X_test)

# 6. Evaluate the model using accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the KNN model: {accuracy * 100:.2f}%')

# Optionally, display the predictions vs. actual values
print(f'Predictions: {y_pred}')
print(f'Actual: {y_test}')
```

```
Accuracy of the KNN model: 100.00%
Predictions: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
Actual: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```

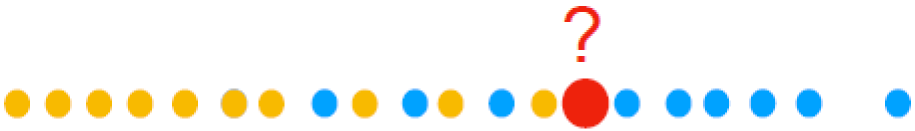
Logistic regression

Classification with logistic regression

Training data: number of hours studied for the course. We also have Pass (1) or Fail (0) label for the data points.

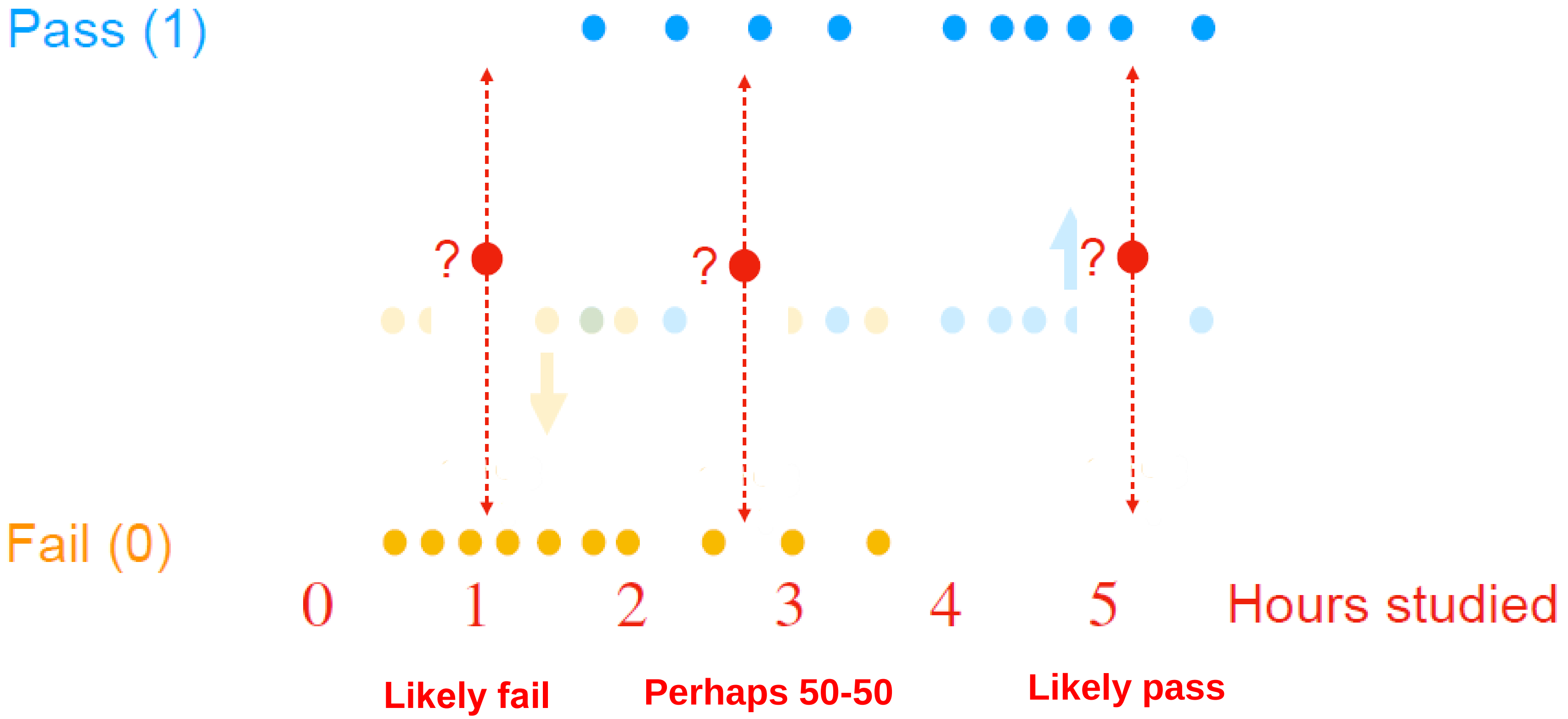
Classification with logistic regression

- Can we train a model so that given a new data point, we can predict whether that student passes or fails?



- Nearest neighbor: a geometric approach for this problem
- We will now approach this question using an alternative probabilistic view..

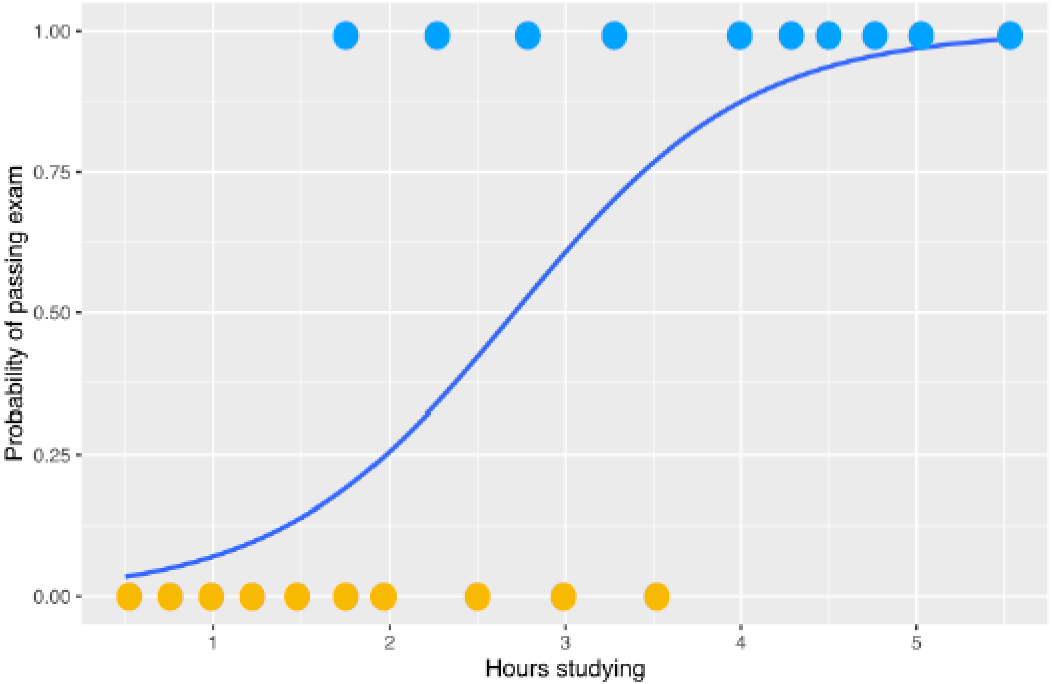
Classification with logistic regression



Classification with logistic regression

$Y = 1$

$Y = 0$



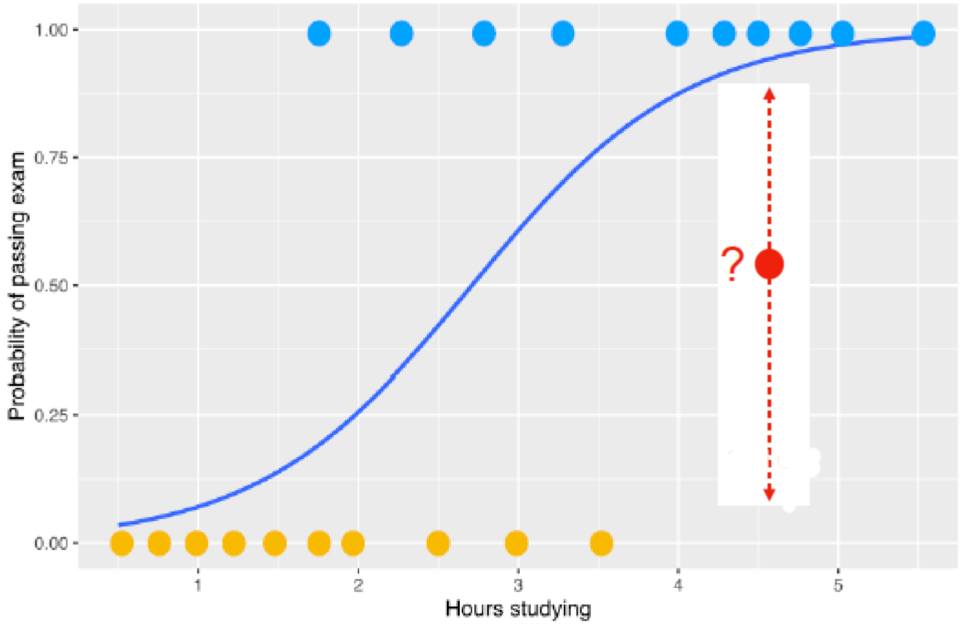
X: feature

Y: label

Blue curve plots:
$P(Y = 1 \mid X = x)$

Classification with logistic regression

$Y = 1$



$Y = 0$

Blue curve plots:

$P(Y = 1 \mid X = x)$

We can predict the class of test point using blue curve:

If prob < 0.5

predict fail

Else

predict pass

How to model the blue curve $P(Y = 1 \mid X = x)$?

Classification with logistic regression

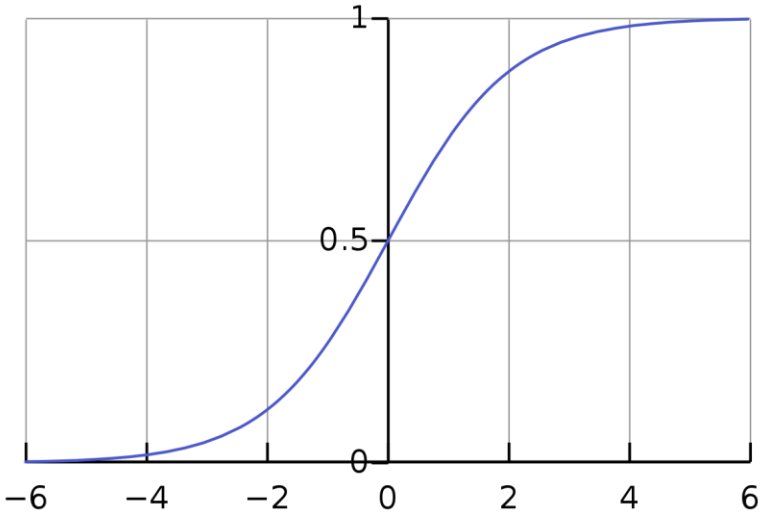
We will model the blue curve with:

$$P(Y = 1 \mid X = x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

For d-dim x, this is dot product

i.e., $\sigma(w \cdot x + b)$,

$\sigma(z) := \frac{1}{1+e^{-z}}$ is the *logistic function*

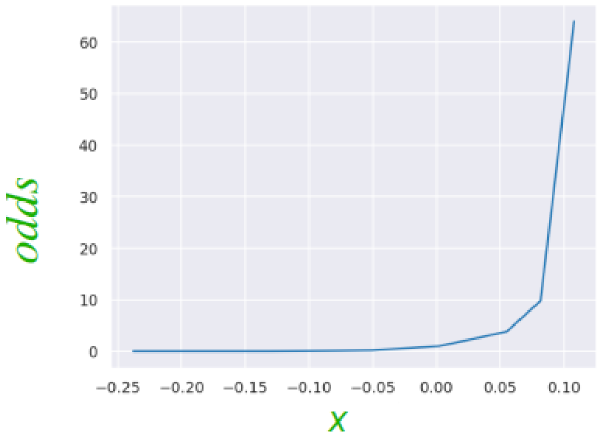


Classification with logistic regression

Where does the logistic function come from?

- Linear regression $w \cdot x + b$ is good at predicting unbounded outputs
- A good unbounded function to predict?

$$\text{odd} = \frac{P(Y=1|x)}{P(Y=-1|x)} = \frac{p}{1-p}$$

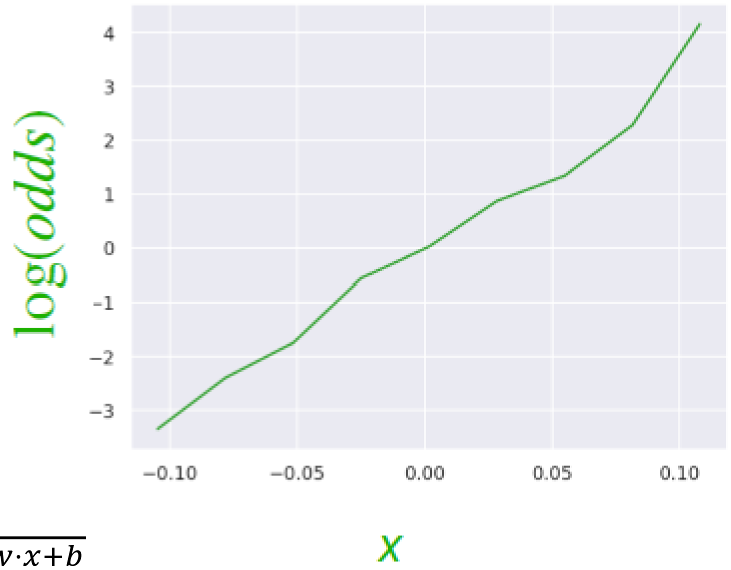


- Still not ideal: odd bounded from below

Classification with logistic regression

Where does the logistic function come from?

- Linear regression $w \cdot x + b$ is good at predicting unbounded outputs
- $\text{log-odd} = \ln \frac{p}{1-p}$
- This now can take +/- values



$$\ln \frac{p}{1-p} = w \cdot x + b \quad \Rightarrow \quad \frac{p}{1-p} = e^{w\,x+b} \quad \Rightarrow \quad p = \frac{1}{1+e^{w \cdot x+b}}$$

Classification with logistic regression

Example Suppose we fit logistic regression model with b = 0.15 and w = 0.575. What is the model's predicted probability that a student who have studied for x = 2 hours passes?

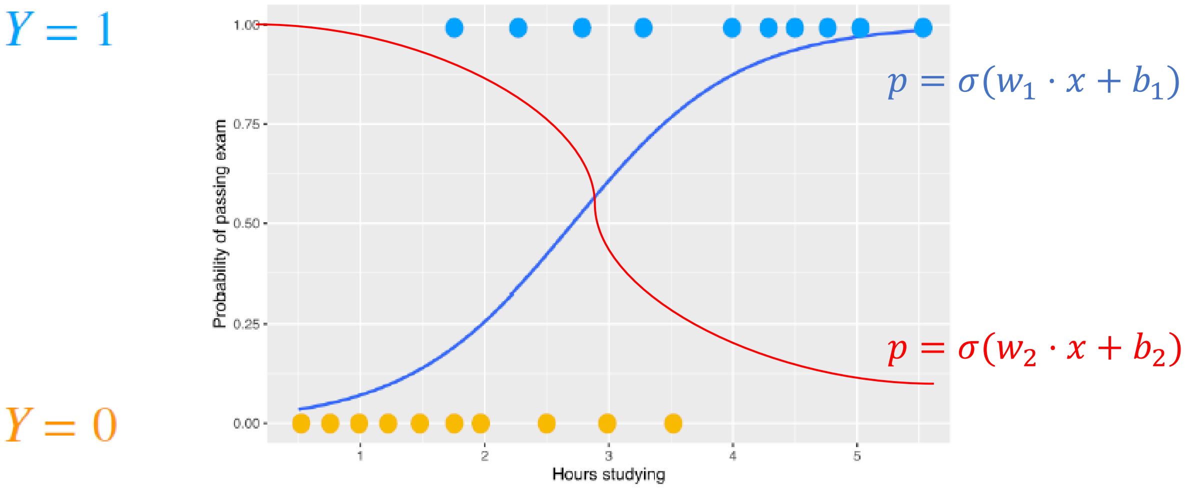
$$P(Y = 1 \mid X = x) = \frac{1}{1+e^{-z}}, \text{ where } z = w \cdot x + b = 1$$

Thus, the predicted pass prob = $\frac{1}{1+e^{-1}} = 0.73$

Fitting a logistic regression model

- Recall: loss for linear regression was MSE $\frac{1}{n}\sum_i (y_i - w \cdot x_i)^2$

- How about logistic regression?
  - y_i's are in 0, 1

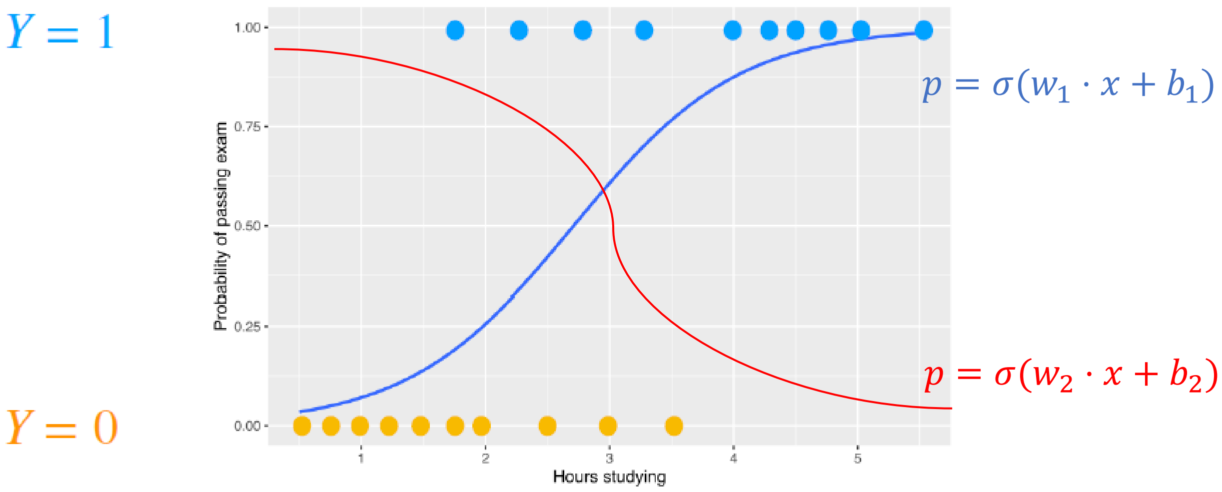


Which logistic regression model fits data better, red or blue?

Fitting a logistic regression model

We'd like to choose w and b such that:

- $w \cdot x + b$, or p, is large for x whose label is more likely to be 1
- $w \cdot x + b$, or p, is small for x whose label is more likely to be 0



Fitting a logistic regression model

- We find w and b to minimize:

$$\sum_i \left(y_i \ln \frac{1}{p_i} + (1 - y_i) \ln \frac{1}{1-p_i} \right),$$ **Cross-entropy (CE) loss**

where $p_i = P(\,Y = 1 \mid x_i\,) = \frac{1}{1+e^{w \cdot x_i + b}}$

- What is the loss when:
  - $y_i = 1$ and $p_i \approx 1$? ≈ 0
  - $y_i = 1$ and $p_i \approx 0$? Large
  - $y_i = 0$ and $p_i \approx 1$? Large
  - $y_i = 0$ and $p_i \approx 0$? ≈ 0

Minimizing CE loss incentivizes the model's predictive probability to align with labels

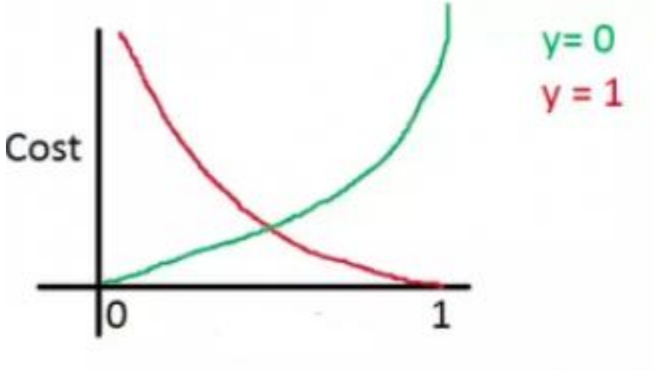
Cross entropy loss

- CE loss:

$$\ell(y,p) = y \ln \frac{1}{p} + (1-y) \ln \frac{1}{1-p}$$

alternative form

$$= \begin{cases} \ln \frac{1}{p}, \ \ y = 1 \\ \ln \frac{1}{1-p}, y = 0 \end{cases}$$



Minimizing CE loss incentivizes the model's predictive probability to align with labels

`sklearn.linear_model`.LogisticRegression

class sklearn.linear_model.LogisticRegression(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) ¶ [source]

penalty : *{'l1', 'l2', 'elasticnet', 'none'}, default='l2'*

Specify the norm of the penalty:

- `'none'`: no penalty is added;
- `'l2'`: add a L2 penalty term and it is the default choice;
- `'l1'`: add a L1 penalty term;
- `'elasticnet'`: both L1 and L2 penalty terms are added.

Similar to linear regression, oftentimes good to add regularization to combat overfitting

tol : *float, default=1e-4*

Tolerance for stopping criteria.

C : *float, default=1.0*

$C = 1/\lambda$

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

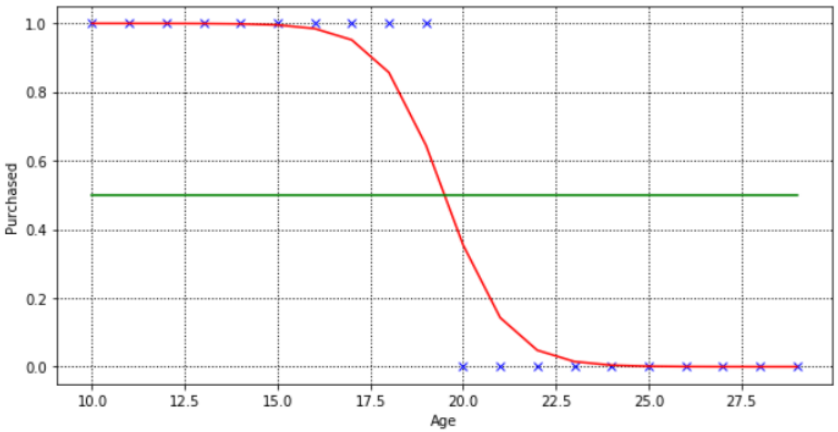
Scikit-Learn Logistic Regression

log_regression = sklearn.linear_model.LogisticRegression()

```
_ = log_regression.fit(pd.DataFrame(x), y)

y_pred = log_regression.predict_proba(pd.DataFrame(x))
log_y_pred_1 = [item[1] for item in y_pred]

fig = plt.figure(figsize=(10,5))
xlabel = 'Age'
ylabel = 'Purchased'
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.grid(color='k', linestyle=':', linewidth=1)
plt.plot(x, y, 'xb')
plt.plot(x, log_y_pred_1, '-r')
_ = plt.plot(x, line_point_5,'-g')
```



Function `predict_proba(X)` returns prediction of class assignment probabilities for each class. It returns n by C matrix if n data points were provided as argument.

(C=number classes)

https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Using Logistic Regression

Logistic Regression have two main usages

- building **predictive** classification models
- **understanding** how features relate to data classes / categories

Example South African Heart Disease (Hastie et al. 2001)
Data result from Coronary Risk-Factor Study in 3 rural areas of South Africa. Data are from white men 15-64yrs. Label is presence/absence of *myocardial infraction (MI).*

Example: African Heart Disease

	sbp	tobacco	ldl	famhist	obesity	alcohol	age	chd
0	160	12.00	5.73	1	25.30	97.20	52	1
1	144	0.01	4.41	0	28.87	2.06	63	1
2	118	0.08	3.48	1	29.14	3.81	46	0
3	170	7.50	6.41	1	31.99	24.26	58	1
4	134	13.60	3.50	1	25.99	57.34	49	1

Features

- Systolic blood pressure
- Tobacco use
- Low density lipoprotein (ldl)
- Family history (discrete)
- Obesity
- Alcohol use
- Age

Q: How predictive is each of the features to myocardial infraction?

Looking at Data

Each scatterplot shows pair of risk factors.
Cases **with** MI (**red**) and **without** (**cyan**)

Features

- Systolic blood pressure
- Tobacco use
- Low density lipoprotein (ldl)
- Family history (discrete)
- Obesity
- Alcohol use
- Age

[Source: Hastie et al. (2001)]

Example: African Heart Disease

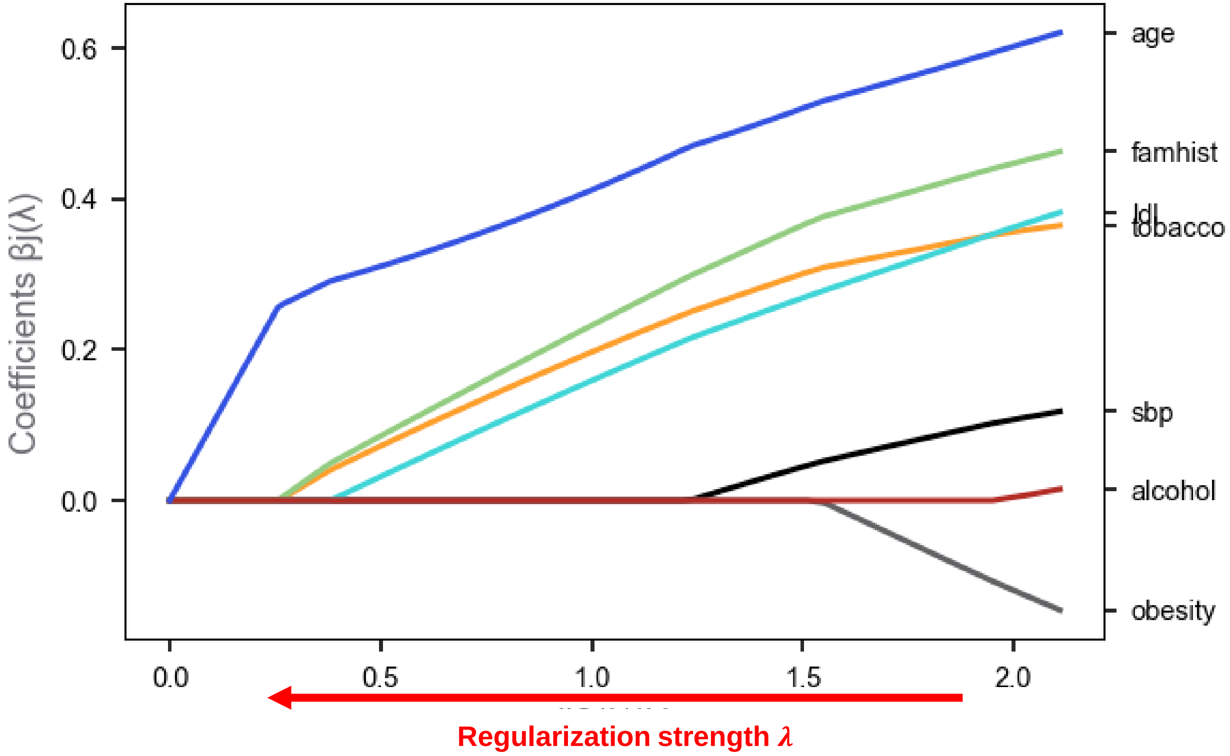
	Coefficient	Std. Error	Z Score
(Intercept)	−4.130	0.964	−4.285
sbp	0.006	0.006	1.023
tobacco	0.080	0.026	3.034
ldl	0.185	0.057	3.219
famhist	0.939	0.225	4.178
obesity	-0.035	0.029	−1.187
alcohol	0.001	0.004	0.136
age	0.043	0.010	4.184

Finding Systolic blood pressure (sbp) and alcohol are **not significant predictors**

Obesity is not significant and negatively correlated with heart disease in the model

Note All correlations / significance of features are based on presence of *other features*. We must always consider that features are strongly correlated.

L1 regularized logistic regression coefficients



https://github.com/empathy87/The-Elements-of-Statistical-Learning-Python-Notebooks/blob/master/examples/South%20African%20Heart%20Disease.ipynb

Classification with logistic regression

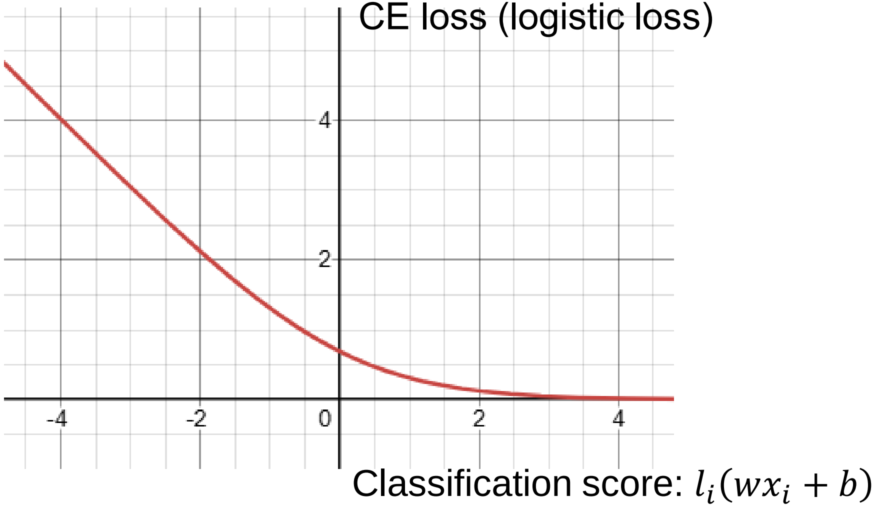
With some algebra, and by redefining our labels as

- $l_i = 1$ if $y_i = 1$
- $l_i = -1$ if $y_i = 0$

Our CE loss can also be written as:

$$\sum_i \ln\left(1 + e^{-l_i(wx_i+b)}\right)$$

$\ln(1 + e^{-z})$: aka the logistic loss



Backup