CS M148 –

Data Science Fundamentals

Lecture #10: Multi-class Logistic Regression & SVM

Baharan Mirzasoleiman

UCLA Computer Science

Announcements

Project 2

- Due Wed Feb 9, 2pm

HW 2

- Will be posted on Wed Feb 9

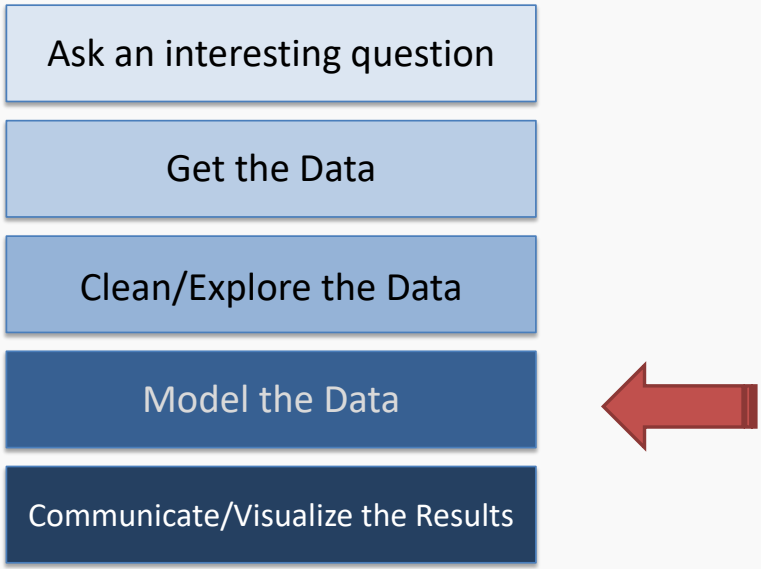
Lecture and discussions

- Online this week
- **Back to in person from Feb 14**

Let’s quickly review what we saw last time

Still here!

The Data Science Process



Classification

Lecture Outline

- Review: Logistic Regression & interpretation
  - Multiple logistic regression
  - Classification boundaries
  - Regularization in logistic regression
- Bayes theorem, Misclassification rates
- Multiclass Classification
  - Multinomial logistic regression
  - OvR logistic regression
- Support Vector Machines
  - Classifying linear separable/non-separable data
  - Kernel trick

Lecture Outline

- Multiclass classification
  - Multinomial logistic regression
  - OvR logistic regression

Heart Data

response variable ***Y*** is Yes/No

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Logistic Regression

Logistic Regression addresses the problem of estimating a probability, $P(y = 1)$, to be outside the range of $[0,1]$. The logistic regression model uses a function, called the ***logistic*** function, to model $P(y = 1)$:

$$P(Y = 1) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Logistic Regression: interpretation

With a little bit of algebraic work, the logistic model can be rewritten as:

$$\ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X.$$

The value inside the natural log function $\frac{P(Y=1)}{1-P(Y=1)}$, is called the ***odds***, thus logistic regression is said to model the ***log-odds*** with a linear function of the predictors or features, X. This gives us the natural interpretation of the estimates similar to linear regression: a one unit change in X is associated with a β_1 change in the log-odds of *success* $(Y = 1)$; or better yet, **a one unit change in $\boldsymbol{X}$ is associated with an $\boldsymbol{e^{\beta_1}}$ multiplicate change in the odds of *success* $(\boldsymbol{Y = 1})$** .

Likelihood

The likelihood of a single observation for *p* given *x and y* is:

$$L(p_i|Y_i) = P(Y_i = y_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

Given the observations are independent, what is the likelihood function for *p*?

$$L(p|Y) = \prod_i P(Y_i = y_i) = \prod_i p_i^{y_i}(1-p_i)^{1-y_i}$$

$$l(p|Y) = -\log L(p|Y) = -\sum_i y_i \log p_i + (1-y_i)\log(1-p_i)$$

Heart Data: logistic estimation

There are various ways to fit a logistic model to this data set in Python. The most straightforward in `sklearn` is via `linear_model.LogisticRegression`.

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(C=100000, fit_intercept=True)
logreg.fit(data_x.values.reshape(-1,1), data_y);


print('Estimated beta1: \n', logreg.coef_)
print('Estimated beta0: \n', logreg.intercept_)
```

```
Estimated beta1:
 [[-0.04326016]]
Estimated beta0:
 [ 6.30193148]
```

Heart Data: logistic estimation

Answer some questions:

- Write down the logistic regression model. $\ln\left(\frac{P(Y=1)}{P(Y=0)}\right) = 6.3 - 0.04\text{MaxHR}$
- Interpret $\hat{\beta}_1$.

  1 unit increase in MaxHR makes the odds of having a hearth disease $e^{-0.04} = 0.96$ times (4% less)

- Estimate the probability of heart disease for someone with MaxHR≈200? $P(Y=1) = \frac{1}{1+e^{-(6.3-0.04MaxHR)}} = 0.15$

Categorical Predictors

Just like in linear regression, when the predictor, X, is binary, the interpretation of the model simplifies (and there is a quick closed form solution here).

In this case, what are the interpretations of $\hat{\beta}_0$ and $\hat{\beta}_1$?

e^{β_0}: odds for group X=0, e^{β_1}: ratio of odds for group X=1 over that of group X=0

For the heart data, let X be the indicator that the individual is a female (X = 0) or male (X = 1). What is the interpretation of the coefficient estimates in this case?

e^{β_0}: odds of HD for women, e^{β_1}: ratio of odds of HD for men over that of women

The observed percentage of HD for women is 26% while it is 55% for men.

Calculate the estimates for $\hat{\beta}_0$ and $\hat{\beta}_1$ if the indicator for HD was predicted from the gender indicator (work on next page).

Solving for the estimates mathematically:

$$P(Y = 1|X = 0) = 0.26, \qquad P(Y = 1|X = 1) = 0.55$$

$$\ln\left(\frac{P(Y = 1)}{1 - P(Y = 1)}\right) = \beta_0 + \beta_1 X$$

$\ln\left(\frac{P(Y=1|X=0)}{P(Y=0|X=0)}\right) = \ln\left(\frac{0.26}{0.74}\right) = -1.04 = \beta_0$ $e^{-1.04} = 0.35$: Odds of having hearth disease for females

$\ln\left(\frac{P(Y = 1|X = 1)}{P(Y = 0|X = 1)}\right) = \ln\left(\frac{0.55}{0.45}\right) = 0.2 = \beta_0 + \beta_1$ $e^{0.2} = 1.22$: Odds of having hearth disease for males

$\beta_1 = 1.24$: Males have $e^{1.24} = 3.45$ times (245% more) odds of having hearth disease

Using Logistic Regression for Classification

How can we use a logistic regression model to perform classification?

That is, how can we predict when $Y = 1$ vs. when $Y = 0$?

We mentioned before, we can classify all observations for which $\hat{P}(Y = 1) \geq 0.5$ to be in the group associated with $Y = 1$ and then classify all observations for which $\hat{P}(Y = 1) < 0.5$ to be in the group associated with $Y = 0$.

Using such an approach is called the standard **Bayes classifier**.

The Bayes classifier takes the approach that assigns each observation to the most likely class, given its predictor values.

Multiple Logistic Regression

Multiple Logistic Regression

Earlier we saw the general form of *simple* logistic regression, meaning when there is just one predictor used in the model. What was the model statement (in terms of linear predictors)?

$$\ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X$$

Multiple logistic regression is a generalization to multiple predictors. More specifically we can define a multiple logistic regression model to predict $P(Y=1)$ as such:

$$\ln\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p$$

Interpretation of Multiple Logistic Regression

Interpreting the coefficients in a multiple logistic regression is similar to that of linear regression.

Key: since there are other predictors in the model, the coefficient $\hat{\beta}_j$ is the association between the j^{th} predictor and the response (on log odds scale). But what do we have to say? Holding all the other predictors fixed, ….

We are trying to attribute the partial effects of each predictor controlling for the others (aka, controlling for possible *confounders*).

Interpreting Multiple Logistic Regression: an Example

Let's get back to the Heart Data. We are attempting to predict whether someone has HD based on MaxHR and whether the person is female or male. The simultaneous effect of these two predictors can be brought into one model.

Recall from earlier we had the following estimated models:

$$\ln\left(\frac{\widehat{P(Y=1)}}{1-\widehat{P(Y=1)}}\right) = 6.30 - 0.043 \cdot X_{MaxHR}$$

$$\ln\left(\frac{\widehat{P(Y=1)}}{1-\widehat{P(Y=1)}}\right) = -1.06 + 1.27 \cdot X_{gender}$$

Interpreting Multiple Logistic Regression: an Example

The results for the multiple logistic regression model are:

```
data_x = df_heart[['MaxHR','Sex']]
data_y = df_heart['AHD']

logreg = LogisticRegression(C=100000, fit_intercept=True)
logreg.fit(data_x, data_y);


print('Estimated beta1: \n', logreg.coef_)
print('Estimated beta0: \n', logreg.intercept_)
```

```
Estimated beta1:
 [[-0.04496354  1.40079047]]
Estimated beta0:
 [ 5.58662464]
```

Some questions

$$\ln\left(\frac{P(Y=1)}{P(Y=0)}\right) = 5.58 - 0.04MaxHR + 1.4Sex$$

1. Estimate the odds ratio of HD comparing men to women using this model.

Keeping MaxHR fixed, men have $e^{1.4} = 4$ times odds of HD compared to women

2. Is there any evidence of multicollinearity in this model?

Yes, the coefficients changed when using both MaxHR and Sex in a model

3. Is there any confounding in this problem?

There might be other factors that are causing this result, e.g. age, exercise, smoking, etc.

Interactions in Multiple Logistic Regression

Just like in linear regression, interaction terms can be considered in logistic regression. An **interaction terms** is incorporated into the model the same way, and the interpretation is very similar (on the log-odds scale of the response of course).

Write down the model for the Heart data for the 2 predictors plus the interactions term based on the output on the next slide.

Interpreting Multiple Logistic Regression: an Example

The results for the multiple logistic regression model are:

```
df_heart['Age_Sex'] = df_heart.Age*df_heart.Sex

data_x = df_heart[['Age','Sex','Age_Sex']]

logit3 = LogisticRegression(penalty='none', max_iter = 1000)
logit3.fit(data_x, df_heart['AHD'])

print("Logistic Regression Estimated Betas (B0,B1,B2,B3):",
      np.round(logit3.intercept_,4),np.round(logit3.coef_,5))
```

```
Logistic Regression Estimated Betas (B0,B1,B2,B3): [-4.2742] [[0.05654 0.77549 0.01273]]
```

$$\ln\left(\frac{P(Y=1)}{P(Y=0)}\right) = -4.27 + 0.05Age + 0.77Sex + 0.01Age\_Sex$$

Some questions

```
Logistic Regression Estimated Betas (B0,B1,B2,B3): [-4.2742] [[0.05654 0.77549 0.01273]]
```

1. Write down the complete model. Break this down into the model to predict log-odds of heart disease (HD) based on Age for women and the same model for men.

$$\ln\left(\frac{P(Y=1|X=0)}{P(Y=0|X=0)}\right) = -4.27 + 0.05Age$$

$$\ln\left(\frac{P(Y=1|X=1)}{P(Y=0|X=1)}\right) = -4.27 + 0.05Age + 0.77 + 0.01Age = -3.5 + 0.06Age$$

Some questions

```
Logistic Regression Estimated Betas (B0,B1,B2,B3): [2.90173899] [[-0.0265903   5.41531302 -0.02707269]]
```

2. Interpret the results of this model. What does the coefficient for the interaction term represent?

- For males, the odds ratio is $e^{0.06} = 1.06$ for a one-unit increase in age,
- For females, the odds ratio for females is $e^{0.05} = 1.05$ for a one-unit increase in age,
- The ratio of these two odds ratios (male over female) is the exponentiated coefficient for the interaction term of **Age_ Sex**: 1.06/1.05 = $e^{0.01}$.

2. Estimate the odds ratio of HD comparing men to women using this model [trick question].

Age	Odds ratio of HD for males over females
50	$\frac{e^{-3.5+0.06\times50}}{e^{-4.27+0.05\times50}} = 3.56$
55	$\frac{e^{-3.5+0.06\times55}}{e^{-4.27+0.05\times55}} = 3.74$
60	$\frac{e^{-3.5+0.06\times60}}{e^{-4.27+0.05\times60}} = 3.93$

Classification Boundaries

Classification boundaries

Recall that we could attempt to purely classify each observation based on whether the estimated $P(Y = 1)$ from the model was greater than 0.5.

Recall, the logistic regression model statement:

$$\ln\left(\frac{P(Y=1)}{P(Y=0)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

If $P(Y = 1) = 0.5$, then the odds is 1, and the log-odds is 0. Thus the classification boundary that separates the estimated probabilities above 0.5 from below 0.5 is defined when we set: $\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = 0$.

Thus logistic regression is said to lead to a **linear classification boundary.** There is actually a lot more freedom in logistic regression's classification boundary (the geometry is defined by the parameterization of your predictors!)

Classification Boundary: a complicated example

```
Logistic Regression Estimated Betas (B0,B1,B2,B3): [-4.2742] [[0.05654 0.77549 0.01273]]
```

4. Using the results of the model above, at what ages will males be predicted to have a heart attack in a classification? At what ages will females be predicted to have a heart attack?

$$\ln\left(\frac{P(Y=1|X=0)}{P(Y=0|X=0)}\right) = -4.27 + 0.05Age = 0$$ Age=85.4

$$\ln\left(\frac{P(Y=1|X=0)}{P(Y=0|X=0)}\right) = -3.5 + 0.06Age = 0$$ Age=58.3

Regularization in Logistic Regression

Regularization in Logistic Regression

A penalty factor can then be added to this loss function and results in a new loss function that penalizes large values of the parameters:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\operatorname{argmin}} \left[-\sum_{i=1}^{n} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

The result is just like in linear regression: shrink the parameter estimates towards zero.

In practice, the intercept is usually not part of the penalty factor.

Note: the sklearn package uses a different tuning parameter: instead of λ they use a constant that is essentially $C = \frac{1}{\lambda}$.

Bayes Theorem, Misclassification Rates, False Positives and Negatives

Bayes' Theorem

We defined conditional probability as:

$$P(B|A) = P(B \text{ and } A)/P(A)$$

And using the fact that $P(B \text{ and } A) = P(A|B)P(B)$ we get the simplest form of Bayes' Theorem:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Another version of Bayes' Theorem is found by substituting in the Law of Total Probability (LOTP) into the denominator:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A|B)P(B) + P(A|B^C)P(B^C)}$$

Where have we seen Bayes' Theorem before? Why do we care?

Diagnostic Testing

In the diagnostic testing paradigm, one cares about whether the results of a test (like a classification test) matches truth (the true class that observation belongs to). The simplest version of this is trying to detect disease ($D+$ vs. $D-$) based on a diagnostic test ($T+$ vs. $T-$).

Medical examples of this include various screening tests: breast cancer screening through (i) self-examination and (ii) mammography, prostate cancer screening through (iii) PSA tests, and Colo-rectal cancer through (iv) colonoscopies.

These tests are a little controversial because of poor predictive probability of the tests.

Diagnostic Testing (cont.)

Bayes' theorem can be rewritten for diagnostic tests:

$$P(D+|T+) = \frac{P(T+|D+)P(D+)}{P(T+|D+)P(D+) + P(T+|D-)P(D-)}$$

These probability quantities can then be defined as:

- *Sensitivity*: $P(T+|D+)$ — Recall
- *Specificity*: $P(T-|D-)$
- *Prevalence*: $P(D+)$
- *Positive Predictive Value*: $P(D+|T+)$ — Precision
- *Negative Predictive Value*: $P(D-|T-)$

How do positive and negative predictive values relate? Be careful...

Diagnostic Testing

We mentioned that these tests are a little controversial because of their poor predictive probability. When will these tests have poor positive predictive probability?

When the disease is not very prevalent, then the number of 'false positives' will overwhelm the number of true positive. For example, PSA screening for prostate cancer has sensitivity of about 90% and specificity of about 97% for some age groups (men in their fifties), but prevalence is about 0.1%.

What is positive predictive probability for this diagnostic test?

$$P(D+|T+) = \frac{P(T+|D+)P(D+)}{P(T+|D+)P(D+) + P(T+|D-)P(D-)} = \frac{0.9 \times 0.001}{0.9 \times 0.001 + 0.03 \times 0.999} = 0.029 = 2.9\%$$

Why do we care?

As data scientists, why do we care about diagnostic testing from the medical world?

Because classification can be thought of as a diagnostic test. Let $Y_i = k$ be the event that observation *i* truly belongs to category *k*, and let $\hat{Y}_i = k$ the event that we correctly predict it to be in class *k*. Then Bayes' rule states that our *Positive Predictive Value* for classification is:

$$P(Y_i = k|\hat{Y}_i = k) = \frac{P(\hat{Y}_i = k|Y_i = k)P(Y_i = k)}{P(\hat{Y}_i = k|Y_i = k)P(Y_i = k) + P(\hat{Y}_i = k|Y_i \neq k)P(Y_i \neq k)}$$

Thus the probability of a predicted outcome truly being in a specific group depends on what? The proportion of observations in that class!

Error in Classification

There are 2 major types of error in classification problems based on a binary outcome. They are:

False positives: incorrectly predicting $\hat{Y} = 1$ when it truly is in $Y = 0$.

False negative: incorrectly predicting $\hat{Y} = 0$ when it truly is in $Y = 1$.

The results of a classification algorithm are often summarized in two ways: (1) a **confusion matrix**, sometimes called a **contingency table**, or a 2x2 table (more generally (*k* x *k*) table) and (2) a receiver operating characteristics (ROC) curve.

Confusion matrix

When a classification algorithm (like logistic regression) is used, the results can be summarize in a (k x k) table as such:

	Predicted no AHD ($\hat{Y} = 0$)	Predicted AHD ($\hat{Y} = 1$)
Truly no AHD ($Y = 0$)	110 TN	54 FP
Truly AHD ($Y = 1$)	53 FN	86 TP

The table above was a classification based on a logistic regression model to predict AHD based on "3" predictors: X_1 = Age, X_2 = Sex, and X_3 = interaction between Age and Sex.

What are the false positive and false negative rates for this classifier?

$$FPR = \frac{FP}{TN + FP} = \frac{54}{110 + 54} = 0.329$$

$$FNR = \frac{FN}{TP + FN} = \frac{53}{86 + 53} = 0.38$$

Bayes' Classifier Choice

A classifier's error rates can be tuned to modify this table. How?

The choice of the Bayes' classifier level will modify the characteristics of this table.

If we thought is was more important to predict AHD patients correctly (fewer false negatives), what could we do for our Bayes' classifier level?

We could classify instead based on:

$$\hat{P}(Y = 1) > \pi$$

and we could choose π to be some level other than 0.50.

Let's see what the table looks like if π were 0.40 or 0.60 instead. What should happen to the False Positive and False Negative frequencies?

Other Confusion tables

Based on π = 0.4:

	Predicted no AHD ($\hat{Y} = 0$)	Predicted AHD ($\hat{Y} = 1$)
Truly no AHD ($Y = 0$)	93 TN	71 FP
Truly AHD ($Y = 1$)	38 FN	101 TP

What has improved? What has worsened? TP, FN are improved, FP, TN are worsened

Based on π = 0.6:

	Predicted no AHD ($\hat{Y} = 0$)	Predicted AHD ($\hat{Y} = 1$)
Truly no AHD ($Y = 0$)	138 TN	26 FP
Truly AHD ($Y = 1$)	74 FN	65 TP

Which should we choose? Why?

If we thought is was more important to predict AHD patients correctly (fewer false negatives), we should use a smaller π. For fewer false positive, we should use a larger π.

Precision and recall

Precision (positive predictive value): $\frac{TP}{TP+FP}$ ← Positive predictions

- number of positive predictions that actually belong to the positive class

Recall (sensitivity): $\frac{TP}{TP+FN}$ ← Actual positives

- how good a test is at detecting the positives

Accuracy = $\frac{TP+FN}{TP+FP+TN+FN}$ (usually use for balanced datasets)

$$F = \frac{2 \times Precision \times recall}{precision + recall}$$ (usually for imbalanced data

ROC Curves and Area-Under-the-Curve

ROC Curves

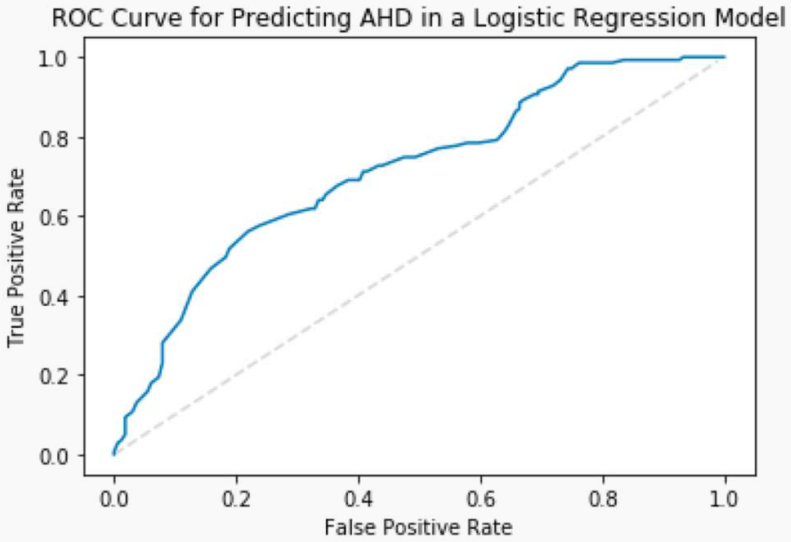
The Radio Operator Characteristics (ROC) curve illustrates the trade-off for all possible thresholds chosen for the two types of error (or correct classification).

The vertical axis displays the true positive predictive value and the horizontal axis depicts the false positive predictive value.

What is the shape of an ideal ROC curve?

See next slide for an example.

ROC Curve Example



AUC for measuring classifier performance

The overall performance of a classifier, calculated over all possible thresholds, is given by the **area under the ROC curve** (AUC).

An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.

What is the worst case scenario for AUC? What is the best case? What is AUC if we independently just flip a [biased] coin to perform classification?

This AUC then can be use to compare various approaches to classification: Logistic regression, *k*-NN, Decision Trees (to come), etc.

Multiclass Classification

Outline: Multiclass Classification

Multinomial Logistic Regression

OvR Logistic Regression

k-NN for multiclass

Outline: Multiclass Classification

Multinomial Logistic Regression

OvR Logistic Regression

k-NN for multiclass

Logistic Regression for predicting 3+ Classes

There are several extensions to standard logistic regression when the response variable *Y* has more than 2 categories. The two most common are:

1. ordinal logistic regression
2. multinomial logistic regression.

Ordinal logistic regression is used when the categories have a specific hierarchy (like class year: Freshman, Sophomore, Junior, Senior; or a 7-point rating scale from strongly disagree to strongly agree).

Multinomial logistic regression is used when the categories have no inherent order (like eye color: blue, green, brown, hazel, et...).

Multiclass Logistic Regression

There are two common approaches to estimating a nominal (not-ordinal) categorical variable that has more than 2 classes. The first approach sets one of the categories in the response variable as the *reference* group, and then fits separate logistic regression models to predict the other cases based off of the reference group. For example we could attempt to predict a student's concentration:

$$y = \begin{cases} 1 & if \text{ Computer Science (CS)} \\ 2 & if \text{ Statistics} \\ 3 & \text{otherwise} \end{cases}$$

from predictors x_1 number of psets per week and x_2 how much time spent in Library.

Multiclass Logistic Regression (cont.)

We could select the $y = 3$ case as the reference group (other concentration), and then fit two separate models: a model to predict $y = 1$ (CS) from $y = 3$ (others) and a separate model to predict $y = 2$ (Stat) from $y = 3$ (others).

Ignoring interactions, how many parameters would need to be estimated?

How could these models be used to estimate the probability of an individual falling in each concentration?

Multinomial Logistic Regression: the model

To predict K classes (K > 2) from a set of predictors X, a multinomial logistic regression can be fit:

$$\ln\left(\frac{P(Y=1)}{P(Y=K)}\right) = \beta_{0,1} + \beta_{1,1}X_1 + \beta_{2,1}X_2 + \cdots + \beta_{p,1}X_p$$
$$\ln\left(\frac{P(Y=2)}{P(Y=K)}\right) = \beta_{0,2} + \beta_{1,2}X_1 + \beta_{2,2}X_2 + \cdots + \beta_{p,2}X_p$$
$$\vdots$$
$$\ln\left(\frac{P(Y=K-1)}{P(Y=K)}\right) = \beta_{0,K-1} + \beta_{1,K-1}X_1 + \beta_{2,K-1}X_2 + \cdots + \beta_{p,K-1}X_p$$

Each separate model can be fit as independent standard logistic regression models!

Multinomial Logistic Regression in sklearn

```
mlogit = LogisticRegression(penalty='none',
                            multi_class='multinomial')
mlogit.fit(nfl_19[['ydstogo','down']],nfl_19['playtype'])
print(mlogit.intercept_)
print(mlogit.coef_)
```

```
[-6.78443446  4.46327069  2.32116377]
[[ 0.08130727  1.86026457]
 [-0.10687815 -1.33128781]
 [ 0.02557089 -0.52897676]]
```

```
pd.crosstab(nfl_19["play_type"],
            nfl_19["playtype"])
```

playtype	0	1	2
play_type			
field_goal	978	0	0
pass	0	0	18981
punt	2150	0	0
qb_kneel	393	0	0
qb_spike	72	0	0
run	0	12994	0

But wait, I thought you said we only fit K – 1 logistic regression models!?!? Why are there K intercepts and K sets of coefficients????

What is sklearn doing?

The K - 1 models in multinomial regression lead to the following probability predictions:

$$\ln\left(\frac{P(Y=k)}{P(Y=K)}\right) = \beta_{0,k} + \beta_{1,k}X_1 + \beta_{2,k}X_k + \cdots + \beta_{p,k}X_p$$

$$\vdots$$

$$P(Y=k) = P(Y=K)e^{\beta_{0,k}+\beta_{1,k}X_1+\beta_{2,k}X_k+\cdots+\beta_{p,k}X_p}$$

This give us K – 1 equations to estimate K probabilities for everyone. But probabilities add up to 1 ☺, so we are all set.

Sklearn then converts the above probabilities back into new betas (just like logistic regression, but the betas won't match):

$$\ln\left(\frac{P(Y=k)}{P(Y\neq k)}\right) = \beta'_{0,k} + \beta'_{1,k}X_1 + \beta'_{2,k}X_k + \cdots + \beta'_{p,k}X_p$$

Outline: Multiclass Classification

Multinomial Logistic Regression

OvR Logistic Regression

k-NN for multiclass

One vs. Rest (ovr) Logistic Regression

An alternative multiclass logistic regression model in sklearn is called the 'One vs. Rest' approach, which is our second method.

If there are 3 classes, then 3 separate logistic regressions are fit, where the probability of each category is predicted over the rest of the categories combined. So for the concentration example, 3 models would be fit:

- a first model would be fit to predict CS from (Stat and Others) combined.
- a second model would be fit to predict Stat from (CS and Others) combined.
- a third model would be fit to predict Others from (CS and Stat) combined.

An example to predict play call from the NFL data follows...

One vs. Rest (OvR) Logistic Regression: the model

To predict K classes (K > 2) from a set of predictors X, a multinomial logistic regression can be fit:

$$\ln\left(\frac{P(Y=1)}{P(Y\neq 1)}\right) = \beta_{0,1} + \beta_{1,1}X_1 + \beta_{2,1}X_2 + \cdots + \beta_{p,1}X_p$$
$$\ln\left(\frac{P(Y=2)}{P(Y\neq 2)}\right) = \beta_{0,2} + \beta_{1,2}X_1 + \beta_{2,2}X_2 + \cdots + \beta_{p,2}X_p$$
$$\vdots$$
$$\ln\left(\frac{P(Y=K)}{P(Y\neq K)}\right) = \beta_{0,K} + \beta_{1,K}X_1 + \beta_{2,K}X_2 + \cdots + \beta_{p,K}X_p$$

Again, each separate model can be fit as independent standard logistic regression models!

Softmax

So how do we convert a set of probability estimates from separate models to one set of probability estimates?

The ***softmax*** function is used. That is, the weights are just normalized for each predicted probability. AKA, predict the 3 class probabilities from each model of the 3 models, and just rescale so they add up to 1.

Mathematically that is:

$$P(y = k|\vec{x}) = \frac{e^{\vec{x}^T \hat{\vec{\beta}}_k}}{\sum_{j=1}^{K} e^{\vec{x}^T \hat{\vec{\beta}}_j}}$$

where $\vec{x}$ is the vector of covariates for that observation and $\hat{\vec{\beta}}_k$ are the associated logistic regression coefficient estimates.

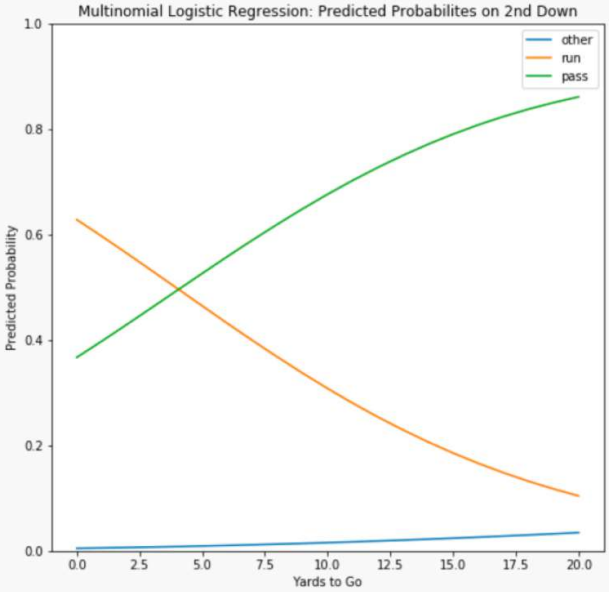
OVR Logistic Regression in Python

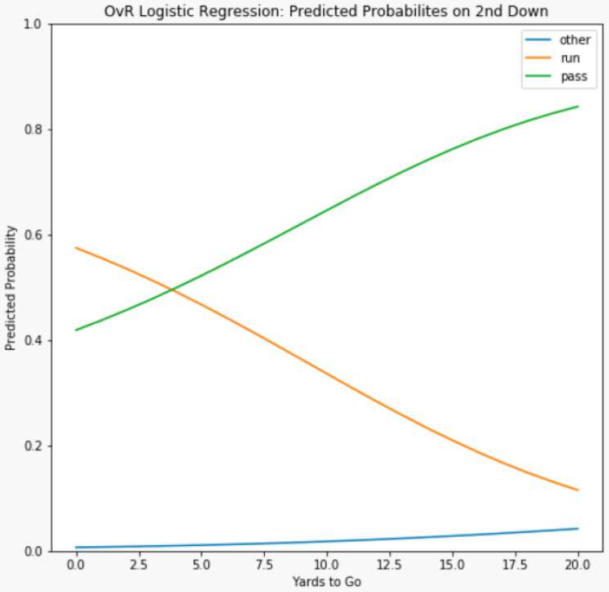
```
ovrlogit = LogisticRegression(penalty='none', multi_class='ovr')

ovrlogit.fit(nfl_19[['ydstogo','down']],nfl_19['playtype'])
print(ovrlogit.intercept_)
print(ovrlogit.coef_)
```

```
[-10.03308293   2.47802369  -0.10976034]
[[ 0.07547391  2.55126265]
 [-0.14272983 -0.98841346]
 [ 0.03672441 -0.03755844]]
```

Phew! This one is as expected ☺

Predicting Type of Play in the NFL





Classification for more than 2 Categories

When there are more than 2 categories in the response variable, then there is no guarantee that $P(Y = k) \geq 0.5$ for any one category. So any classifier based on logistic regression (or other classification model) will instead have to select the group with the **largest estimated probability**.

The classification boundaries are then much more difficult to determine. We will not get into the algorithm for drawing these in this class, but we can rely on `predict` and `predict_proba`!

Prediction using Multiclass Logistic Regression

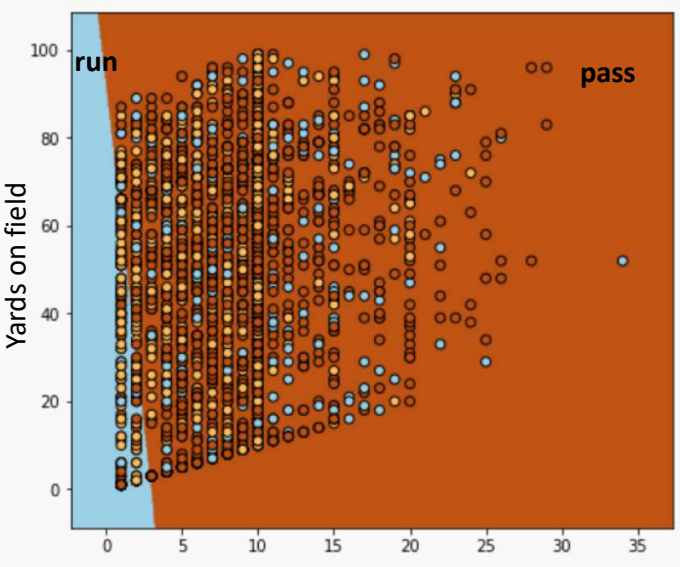
```
print(mlogit.predict_proba(nfl_19[['ydstogo','down']])[0:5,:])
print(mlogit.predict(nfl_19[['ydstogo','down']])[0:5])
```

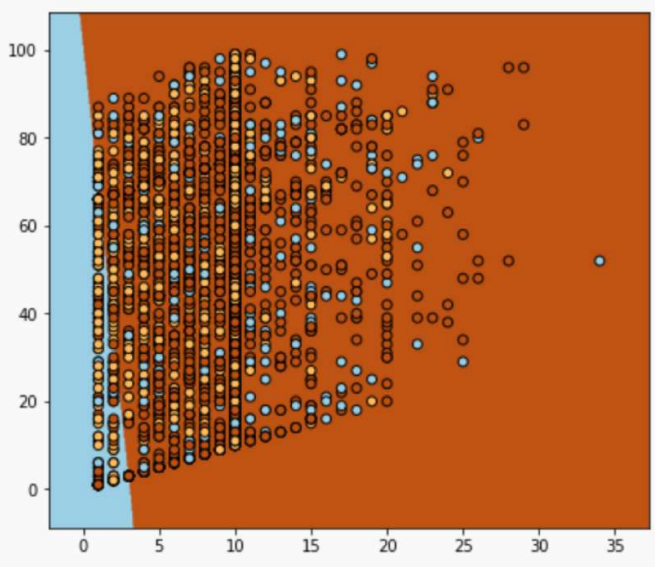
```
[[0.001048   0.50329827 0.49565372]
 [0.01559789 0.30793278 0.67646934]
 [0.17275682 0.14020122 0.68704196]
 [0.82376365 0.00418569 0.17205066]
 [0.001048   0.50329827 0.49565372]]
[1 2 2 0 1]
```

```
print(ovrlogit.predict_proba(nfl_19[['ydstogo','down']])[0:5,:])
print(ovrlogit.predict(nfl_19[['ydstogo','down']])[0:5])
```

```
[[0.00111671 0.48115571 0.51772757]
 [0.01792012 0.33603242 0.64604746]
 [0.19847963 0.15493954 0.64658083]
 [0.57254676 0.0088239  0.41862935]
 [0.00111671 0.48115571 0.51772757]]
[2 2 2 0 2]
```

Classification Boundary for 3+ Classes in sklearn





Estimation and Regularization in multiclass settings

There is no difference in the approach to estimating the coefficients in the multiclass setting: we maximize the log-likelihood (or minimize negative log-likelihood).

This combined negative log-likelihood of all K classes is sometimes called the **binary cross-entropy**:

means if is true, count 1

$$\ell = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(y_i = k)\ln(\hat{P}\,(y_i = k)) + \mathbb{1}(y_i \neq k)\ln(1 - \hat{P}\,(y_i = k))$$

And regularization can be done like always: add on a penalty term to this loss function based on L1 (sum of the absolute values) or L2 (sum of squares) norms.

Outline: Multiclass Classification

Multinomial Logistic Regression

OvR Logistic Regression

k-NN for multiclass

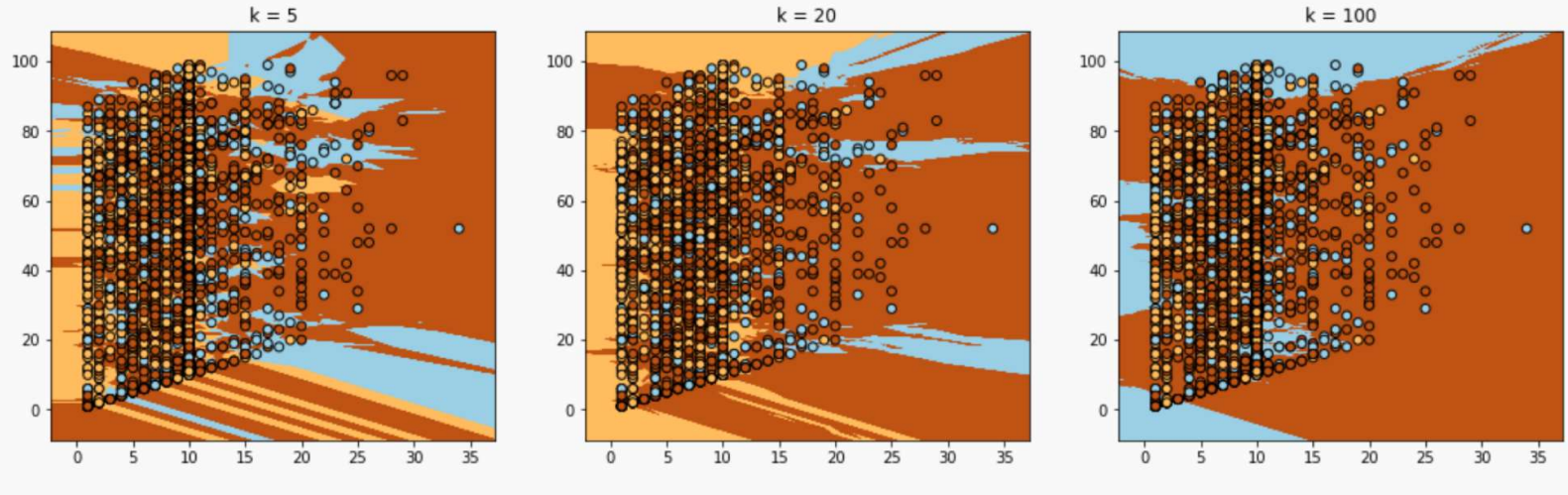
k-NN for 3+ Classes

Extending the k-NN classification model to 3+ classes is much simpler!

Remember, k-NN is done in two steps:

1. **Find you k neighbors**: this is still done in the exact same way
   – be careful of the scaling of your predictors!
2. **Predict based on your neighborhood**:
   – Predicting probabilities: just use the observed proportions
   – Predicting classes: plurality wins!

k-NN for 3+ Classes: NFL Data



Support Vector Machines (SVMs)

Outline

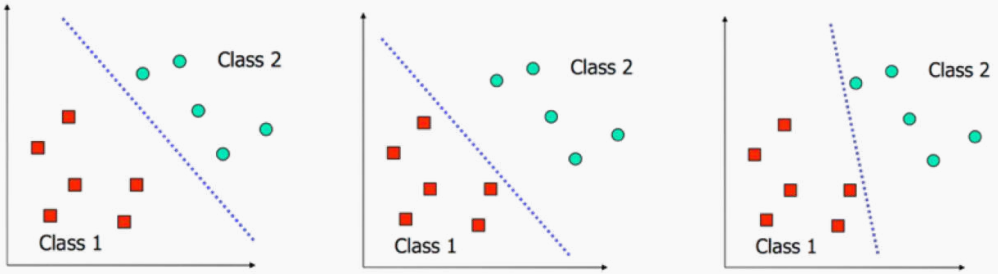
- Classifying Linear Separable Data
- Classifying Linear Non-Separable Data
- Kernel Trick

Decision Boundaries Revisited

In logistic regression, we learn a ***decision boundary*** that separates the training classes in the feature space.

When the data can be perfectly separated by a linear boundary, we call the data ***linearly separable***.

In this case, multiple decision boundaries can fit the data. How do we choose the best?



Question: What happens to our logistic regression model when training on linearly separable datasets?

Decision Boundaries Revisited (cont.)

Constraints on the decision boundary:

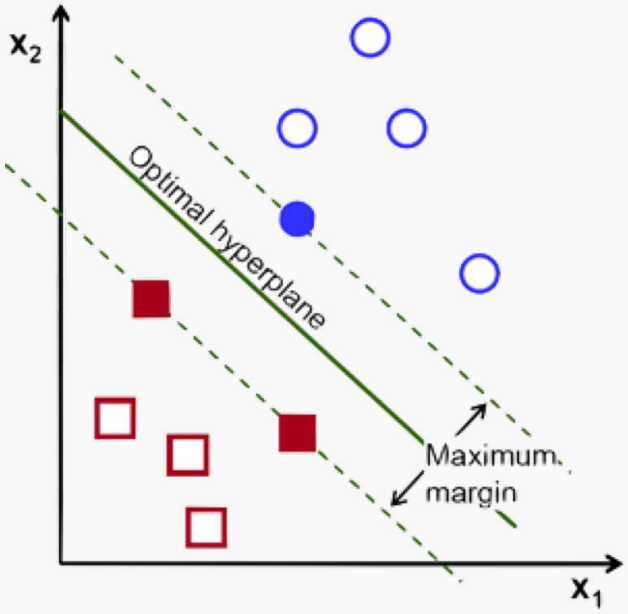
- In logistic regression, we typically learn an $\ell 1$ or $\ell 2$ regularized model.
- So, when the data is linearly separable, we choose a model with the 'smallest coefficients' that still separate the classes.
- The purpose of regularization is to prevent overfitting.

Decision Boundaries Revisited (cont.)

Constraints on the decision boundary:

- We can consider alternative constraints that prevent overfitting.
- For example, we may prefer a decision boundary that does not ‘favor’ any class (esp. when the classes are roughly equally populous).
- Geometrically, this means choosing a boundary that maximizes the distance or ***margin*** between the boundary and both classes.

Illustration of an SVM

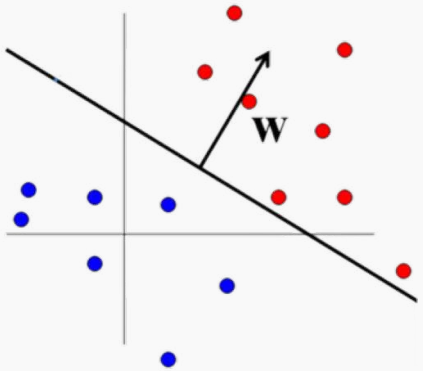


Geometry of Decision Boundaries

Recall that the decision boundary is defined by some equation in terms of the predictors. A linear boundary is defined by:

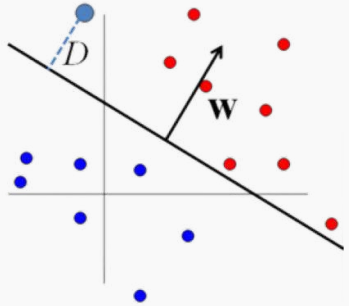
$$w^\top x + b = 0 \text{ (General equation of a hyperplane)}$$

Recall that the non-constant coefficients, w, represent a ***normal vector***, pointing orthogonally away from the plane



Geometry of Decision Boundaries (cont.)

Now, using some geometry, we can compute the distance between any point to the decision boundary using *w* and *b*.



The signed distance from a point $x \in \mathbb{R}^n$ to the decision boundary is

$$D(x) = \frac{w^\top x + b}{\|w\|}$$ (Euclidean Distance Formula)

Maximizing Margins

Now we can formulate our goal - find a decision boundary that maximizes the distance to both classes - as an optimization problem:

$$\begin{cases} \max\limits_{w,b} M \\ \text{such that } |D(x_n)| = \frac{y_i(w^\top x_n + b)}{\|w\|} \geq M, \; n = 1, \ldots, N \end{cases}$$

where M is a real number representing the width of the 'margin' and $y_i = \pm 1$. The inequalities $|D(x_n)| \geq M$ are called ***constraints***.

The constrained optimization problem as present here looks tricky. Let's simplify it with a little geometric intuition.

Maximizing Margins (cont.)

Notice that maximizing the distance of ***all points*** to the decision boundary, is exactly the same as maximizing the distance to the ***closest points***.

The points closest to the decision boundary are called ***support vectors***.

For any plane, we can always scale the equation:

$$w^\top x + b = 0$$

so that the support vectors lie on the planes:

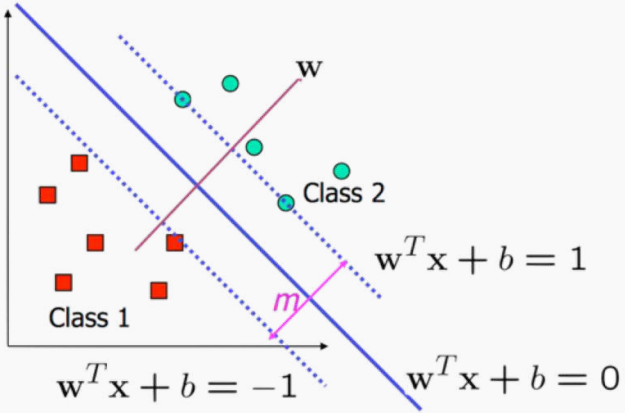
$$w^\top x + b = \pm 1,$$

depending on their classes.

Maximizing Margins Illustration

For points on planes $w^T x + b = \pm 1$, their distance to the decision boundary is $\pm 1/\|w\|$.

So we can define the ***margin*** of a decision boundary as the distance to its support vectors, $m = 2/\|w\|$.



Support Vector Classifier: Hard Margin

Finally, we can reformulate our optimization problem - find a decision boundary that maximizes the distance to both classes - as the maximization of the margin, *m*, ***while maintaining zero misclassifications***,

$$\begin{cases} \max\limits_{w,b} \dfrac{2}{\|w\|} \\ \text{such that } y_n(w^\top x_n + b) \geq 1,\ n = 1, \ldots, N \end{cases}$$

The classifier learned by solving this problem is called ***hard margin support vector classification***.

Often SVC is presented as a minimization problem:

$$\begin{cases} \min\limits_{w,b} \|w\|^2 \\ \text{such that } y_n(w^\top x_n + b) \geq 1,\ n = 1, \ldots, N \end{cases}$$

SVC and Convex Optimization

As a convex optimization problem SVC has been extensively studied and can be solved by a variety of algorithms:

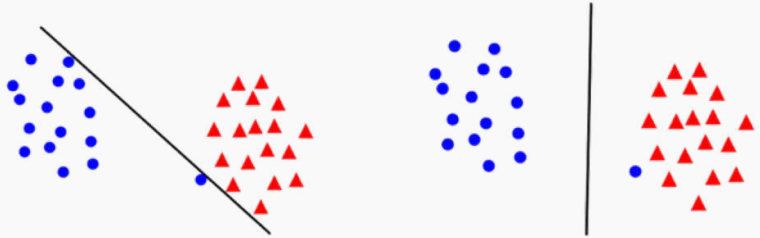
- **(Stochastic)** libLinear
  Fast convergence, moderate computational cost
- **(Greedy)** libSVM
  Fast convergence, moderate computational cost
- **(Stochastic)** Stochastic Gradient Descent Slow convergence, low computational cost per iteration
- **(Greedy)** Quasi-Newton Method
  Very fast convergence, high computational cost

Classifying Linear Non-Separable Data

Geometry of Data

Maximizing the margin is a good idea as long as we assume that the underlying classes are linear separable and that the data is noise free.

If data is noisy, we might be sacrificing generalizability in order to minimize classification error with a very narrow margin:



With every decision boundary, there is a trade-off between maximizing margin and minimizing the error.

Support Vector Classifier: Soft Margin

Since we want to balance maximizing the margin and minimizing the error, we want to use an objective function that takes both into account:

$$\begin{cases} \min\limits_{w,b} \|w\|^2 + \lambda \text{Error}(w,b) \\ \text{such that } y_n(w^\top x_n + b) \geq 1, \ n = 1, \ldots, N \end{cases}$$

where λ is an intensity parameter.

So just how should we compute the error for a given decision boundary?

Support Vector Classifier: Soft Margin (cont.)

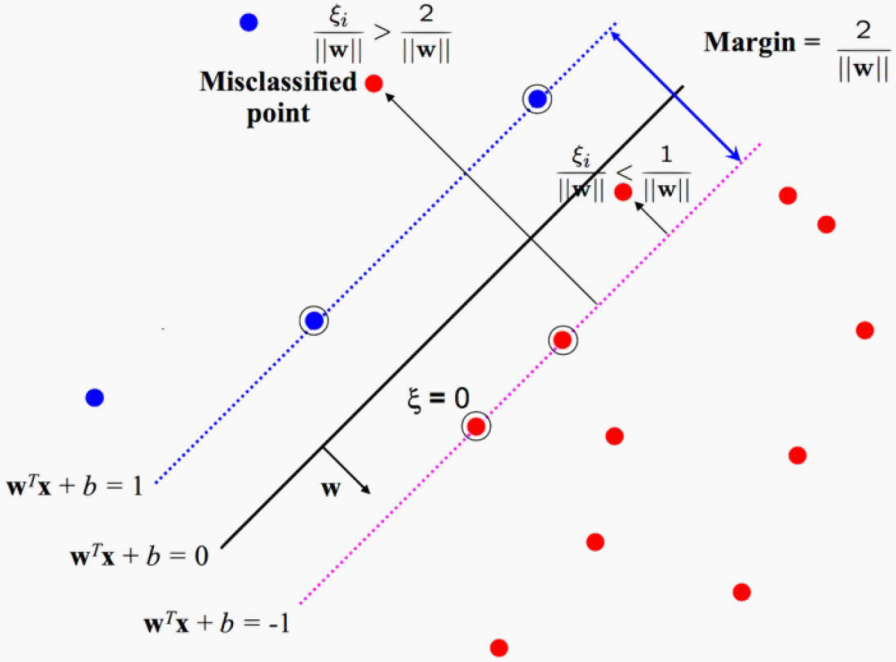
We want to express the error as a function of distance to the decision boundary.

Recall that the support vectors have distance $1/\|w\|$ to the decision boundary. We want to penalize two types of 'errors'

- **(margin violation)** points that are on the correct side of the boundary but are inside the margin. They have distance $\xi/\|w\|$, where $0 < \xi < 1$.
- **(misclassification)** points that are on the wrong side of the boundary. They have distance $\xi/\|w\|$, where $\xi > 1$.

Specifying a nonnegative quantity for ξ_n is equivalent to quantifying the error on the point x_n.

Support Vector Classifier: Soft Margin Illustration



Support Vector Classifier: Soft Margin (cont.)

Formally, we incorporate error terms ξ_n 's into our optimization problem by:

$$\begin{cases} \min\limits_{\xi_n \in \mathbb{R}^+, w, b} \|w\|^2 + \lambda \sum\limits_{n=1}^{N} \xi_n \\ \text{such that } y_n(w^\top x_n + b) \geq 1 - \xi_n, \ n = 1, \ldots, N \end{cases}$$

The solution to this problem is called ***soft margin support vector classification*** or simply ***support vector classification***.

Tuning SVC

Choosing different values for λ in

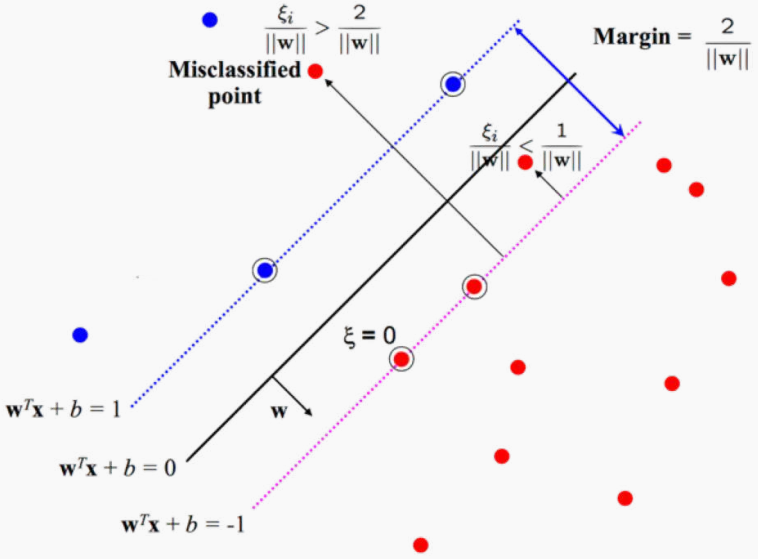
$$\begin{cases} \min\limits_{\xi_n \in \mathbb{R}^+, w, b} \|w\|^2 + \lambda \sum\limits_{n=1}^{N} \xi_n \\ \text{such that } y_n(w^\top x_n + b) \geq 1 - \xi_n, \ n = 1, \ldots, N \end{cases}$$

will give us different classifiers. In general,

- small λ penalizes errors less and hence the classifier will have a large margin
- large λ penalizes errors more and hence the classifier will accept narrow margins to improve classification
- setting $\lambda = \infty$ produces the hard margin solution

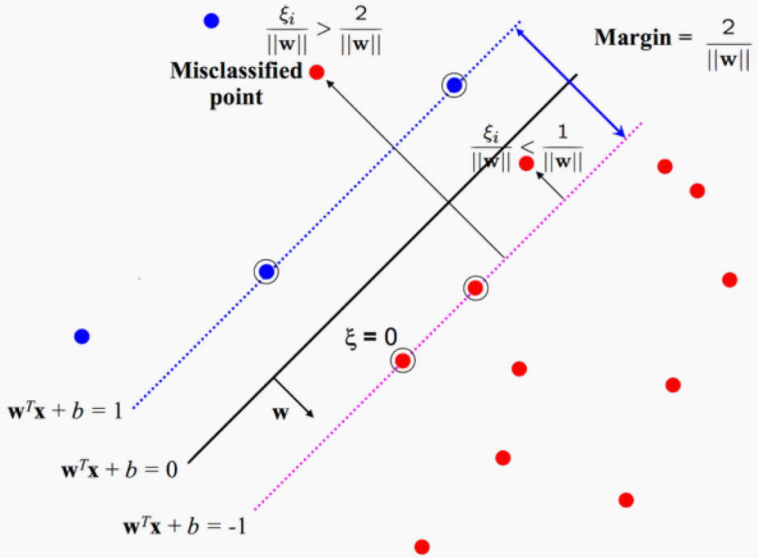
Decision Boundaries and Support Vectors

Recall how the error terms ξ_n's were defined: the points where $\xi_n = 0$ are precisely the support vectors



Decision Boundaries and Support Vectors

Thus to re-construct the decision boundary, ***only the support vectors are needed!***



Decision Boundaries and Support Vectors

The decision boundary of an SVC is given by

$$\hat{w}^\top x + \hat{b} = \sum_{x_n \text{ is a support vector}} \hat{\alpha}_n y_n (x_n^\top x_n) + b$$

where $\hat{\alpha}_n$ and the set of support vectors are found by solving the optimization problem.

- To classify a test point x_{test}, we predict

$$\hat{y}_{test} = \text{sign}\left(\hat{w}^\top x + \hat{b}\right)$$

SVC as Optimization

With the help of geometry, we translated our wish list into an optimization problem

$$\begin{cases} \min\limits_{\xi_n \in \mathbb{R}^+, w, b} \|w\|^2 + \lambda \sum\limits_{n=1}^{N} \xi_n \\ \text{such that } y_n(w^\top x_n + b) \geq 1 - \xi_n, \ n = 1, \ldots, N \end{cases}$$

where ξ_n quantifies the error at x_n.

The SVC optimization problem is often solved in an alternate form (the dual form):

$$\max_{\alpha_n \geq 0, \sum_n \alpha_n y_n = 0} \sum_n \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} y_n y_m \alpha_n \alpha_m x_n^\top x_m$$

Later we'll see that this alternate form allows us to use SVC with non-linear boundaries.

Extension to Non-linear Boundaries

Polynomial Regression: Two Perspectives

Given a training set:

$$\{(x_1, y_1), \ldots, (x_N, y_N)\}$$

with a single real-valued predictor, we can view fitting a 2nd degree polynomial model:

$$w_0 + w_1 x + w_2 x^2$$

on the data as the process of finding the best quadratic curve that fits the data. But in practice, we first expand the feature dimension of the training set

$$x_n \mapsto (x_n^0, x_n^1, x_n^2)$$

and train a ***linear model*** on the expanded data

$$\{(x_n^0, x_n^1, x_N^2, y_1), \ldots, (x_N^0, x_N^1, x_N^2, y_N)\}$$

Transforming the Data

The key observation is that training a polynomial model is just training a linear model on data with transformed predictors.

In our previous example, transforming the data to fit a 2nd degree polynomial model requires a map:

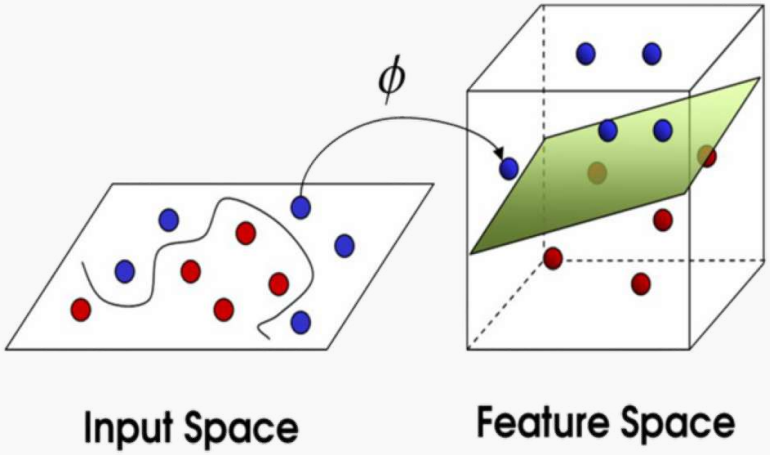
$$\phi : \mathbb{R} \to \mathbb{R}^3$$
$$\phi(x) = (x^0, x^1, x^2)$$

where $\mathbb{R}$ called the ***input space***, $\mathbb{R}^3$ is called the ***feature space***.

While the response may not have a linear correlation in the input space $\mathbb{R}$, it may have one in the feature space $\mathbb{R}^3$.

SVC with Non-Linear Decision Boundaries

The same insight applies to classification: while the response may not be linear separable in the input space, it may be in a feature space after a fancy transformation:



SVC with Non-Linear Decision Boundaries (cont.)

The motto: instead of tweaking the definition of SVC to accommodate non-linear decision boundaries, we map the data into a feature space in which the classes are linearly separable (or nearly separable):

- Apply transform ϕ: $\mathbb{R}^J \rightarrow \mathbb{R}^{J'}$ on training data

  $$x_n \rightarrow \phi(x_n)$$

  where typically *J*′ is much larger than *J*.
- Train an SVC on the transformed data

  $$\{(\phi(x_1), y_1), (\phi(x_2), y_2), \ldots, (\phi(x_N), y_N)\}$$

Inner Products

Since the feature space $\mathbb{R}^{J'}$ is potentially extremely high dimensional, computing ϕ explicitly can be costly.

Instead, we note that computing ϕ is unnecessary. Recall that training an SVC involves solving the optimization problem:

$$\max_{\alpha_n \geq 0,\, \sum_n \alpha_n y_n = 0} \sum_n \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} y_n y_m \alpha_n \alpha_m \phi(x_n)^\top \phi(x_m)$$

In the above, ***we are only interested in computing inner products*** $\phi(x_n)^T \phi(x_m)$***in the feature space*** and not the quantities $\phi(x_n)$ themselves.

The Kernel Trick

The ***inner product*** between two vectors is a measure of the similarity of the two vectors.

Definition

Given a transformation $\phi : \mathbb{R}^J \to \mathbb{R}^{J'}$, from input space $\mathbb{R}^J$ to feature space $\mathbb{R}^{J'}$, the function $K : \mathbb{R}^J \times \mathbb{R}^J \to \mathbb{R}$ defined by

$$K(x_n, x_m) = \phi(x_n)^\top \phi(x_m), \quad x_n, x_m \in \mathbb{R}^J$$

is called the ***kernel function*** of ϕ.

Generally, ***kernel function*** may refer to any function $K : \mathbb{R}^J \times \mathbb{R}^J \to \mathbb{R}$ that measure the similarity of vectors in $\mathbb{R}^J$, without explicitly defining a transform ϕ.

The Kernel Trick (cont.)

For a choice of kernel *K*,

$$K(x_n, x_m) = \phi(x_n)^\top \phi(x_m)$$

we train an SVC by solving

$$\max_{\alpha_n \geq 0,\ \sum_n \alpha_n y_n = 0} \sum_n \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} y_n y_m \alpha_n \alpha_m K(x_n, x_m)$$

Computing $K(x_n, x_m)$ can be done without computing the mappings $\phi(x_n)$, $\phi(x_m)$.

This way of training a SVC in feature space without explicitly working with the mapping ϕ is called ***the kernel trick***.

Transforming Data: An Example

Example

Let's define $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ by

$$\phi([x_1, x_2]) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

The inner product in the feature space is

$$\phi([x_{11}, x_{12}])^\top \phi([x_{21}, x_{22}]) = (1 + x_{11}x_{21} + x_{12}x_{22})^2$$

Thus, we can directly define a kernel function $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ by

$$K(x_1, x_2) = (1 + x_{11}x_{21} + x_{12}x_{22})^2.$$

Notice that we need not compute $\phi([x_{11}, x_{12}])$, $\phi([x_{21}, x_{22}])$ to compute $K(x_1, x_2)$.

Kernel Functions

Common kernel functions include:

- **Polynomial Kernel** (kernel='poly')

$$K(x_1, x_2) = (x_1^\top x_2 + 1)^d$$

  where *d* is a hyperparameter.

- **Radial Basis Function Kernel** (kernel='rbf')

$$K(x_1, x_2) = \exp\left\{-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right\}$$

  where σ is a hyperparameter.

- **Sigmoid Kernel** (kernel='sigmoid')

$$K(x_1, x_2) = \tanh(\kappa x_1^\top x_2 + \theta)$$

  where κ and θ are hyperparameters.