Data Analytics (PETR 6397)

Classification

Dr. Ahmad Sakhaee-Pour

Petroleum Engineering Department

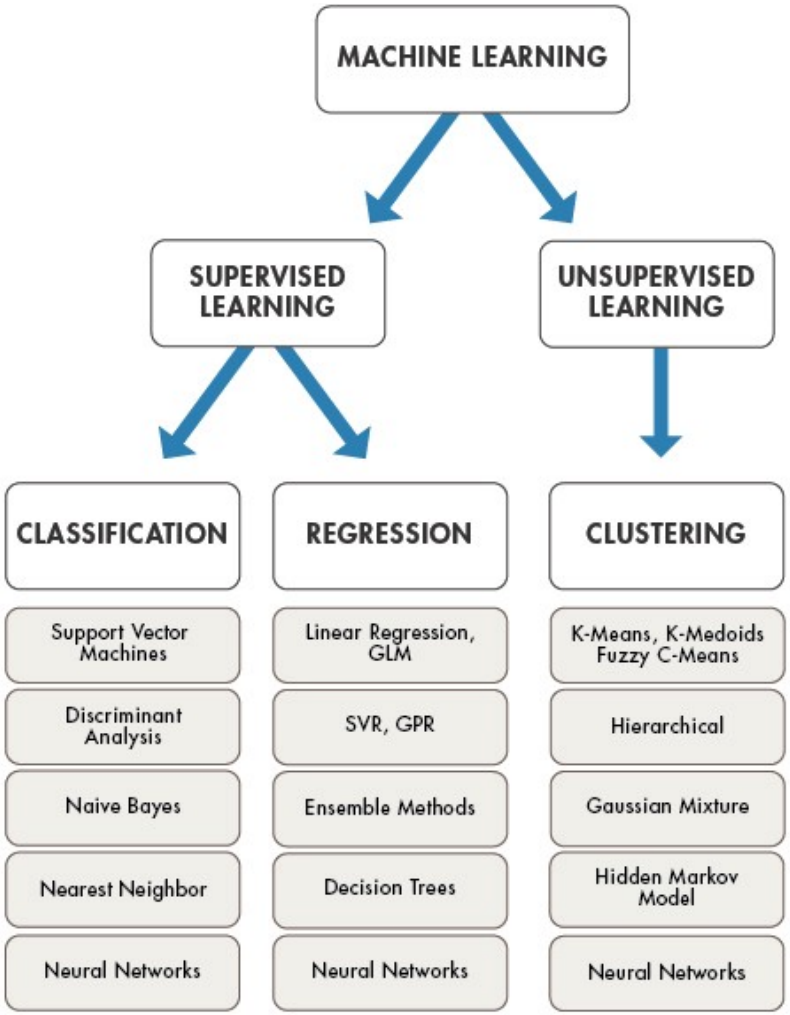
University of Houston

Spring 2025

Discussed topics

- Classification
- Commonly used datasets
- Stochastic Gradient Descent (SGD) classifier
- Classification Metrics
    - Precision and Trade
    - Confusion Matrix (CM)
    - Receiver Operating Characteristics (ROC)
- K-nearest neighbor (KNN)
- Multiclass classification
- Parametric methods (Logistic Regression)
- Random Forest

Reminder of various problems



Classification

- In classification, we predict a class label assigning points to groups
- In classification, the training inputs are similar to regression, but the outputs are discrete values, not continuous
- In classification, we know the correct outputs (which are unknown in clustering)

Classification types

Binary Classification

- Spam
- Not spam

Multiclass Classification

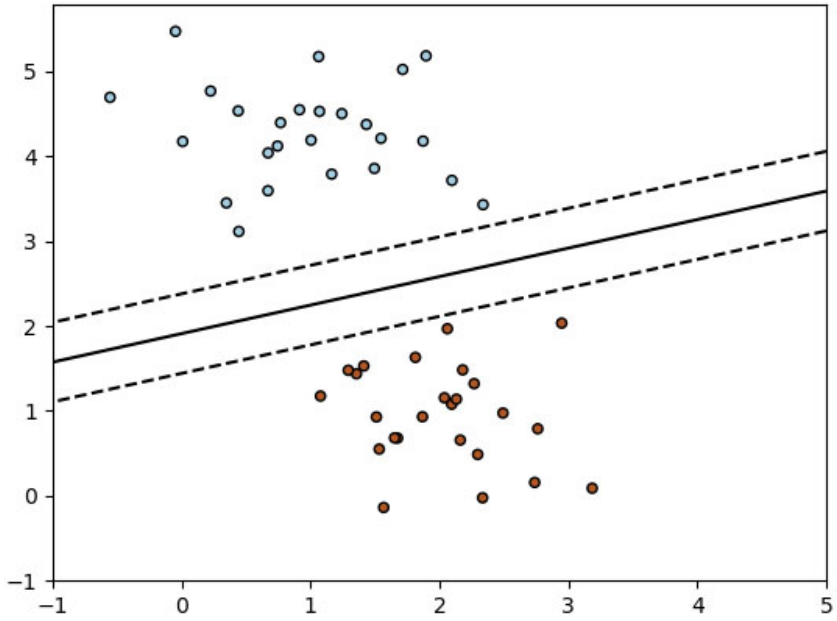
- Dog
- Cat
- Horse
- Fish
- Bird
- ...

Multi-label Classification

- Dog
- Cat
- Horse
- Fish
- Bird
- ...

Stochastic Gradient Descent (SGD) classifier

- It implements a stochastic gradient descent routine that supports different loss functions and penalties for classification

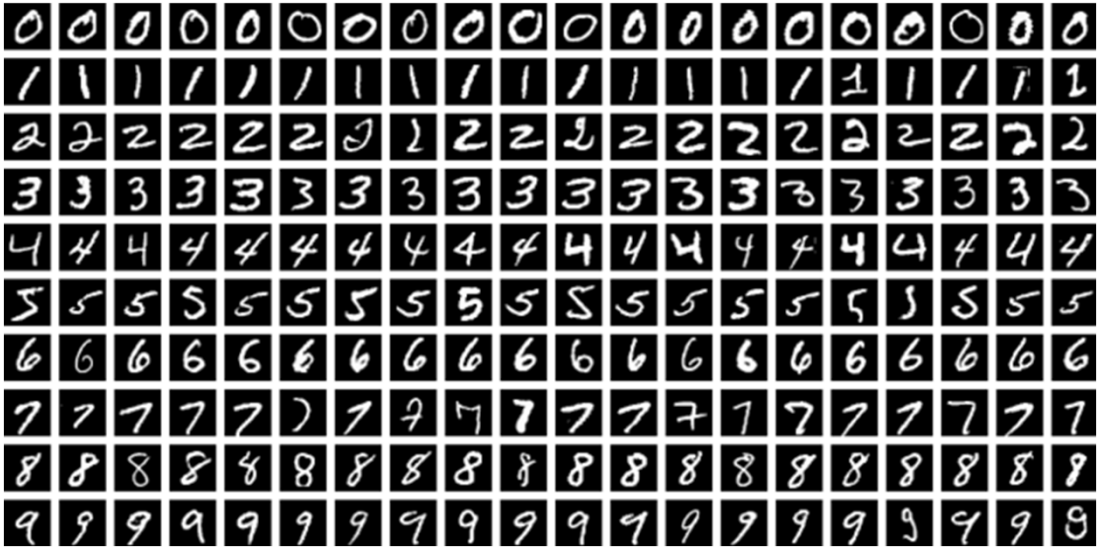


Commonly used datasets

- MNIST: Modified National Institute of Standards and Technology
- Fashion MNIST
- Canadian Institute for Advanced Research: CIFAR-10, CIFAR-100
- CelebFaces Attributes Dataset: CelebA
- …

Modified National Institute of Standards and Technology (MNIST)

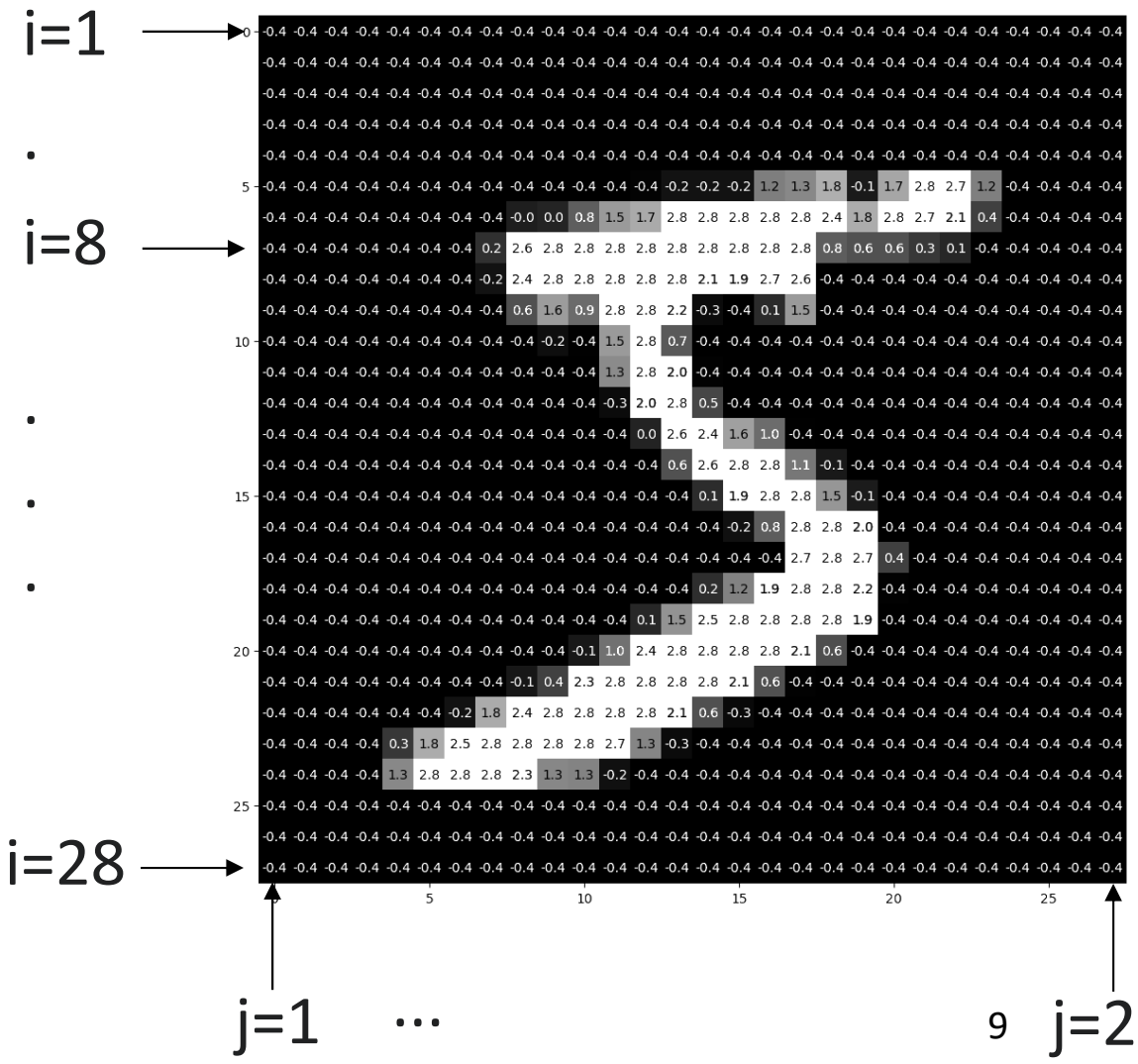
- A large database of handwritten digits from 0 to 9
- It is used for training various models
- The MNIST database has 60,000 training images and 10,000 testing images



Each image has 784 pixels

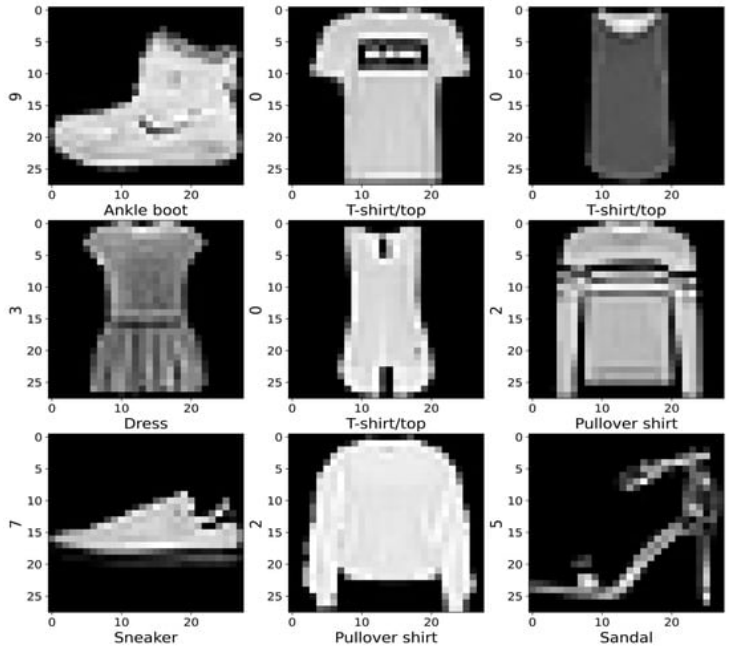
- Each image 28x28 pixels
- Each pixel has a single value that is smaller for the background (black region)
- Majority of pixels are background (black region)

Pixel values are scaled below



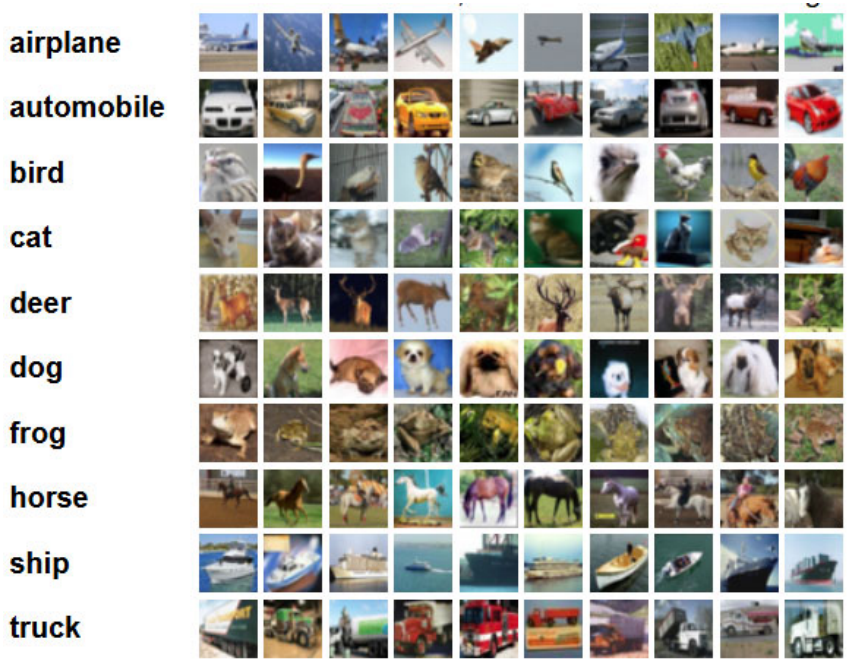
Fashion-MNIST

- It has 60,000 images (50,000 for training and 10,000 for testing)
- Each image has 28x28 pixels
- It has 10 classes



CIFAR-10

- 60000 color images in 10 classes
- Each image has 32x32 pixels
- The images are divided into 50000 training and 10000 test images



- Q: What does CIFAR-100 have?

Building a classifier

- Step 1: Split the data into training and testing
- Step 2: Select the function (classifier model)
- Step 3: Train the classifier on the training data
- Step 4a: Evaluate the classifier on the training data (memorization performance)
- Step 4b: Evaluate the trained classifier on the testing data (generalization performance)

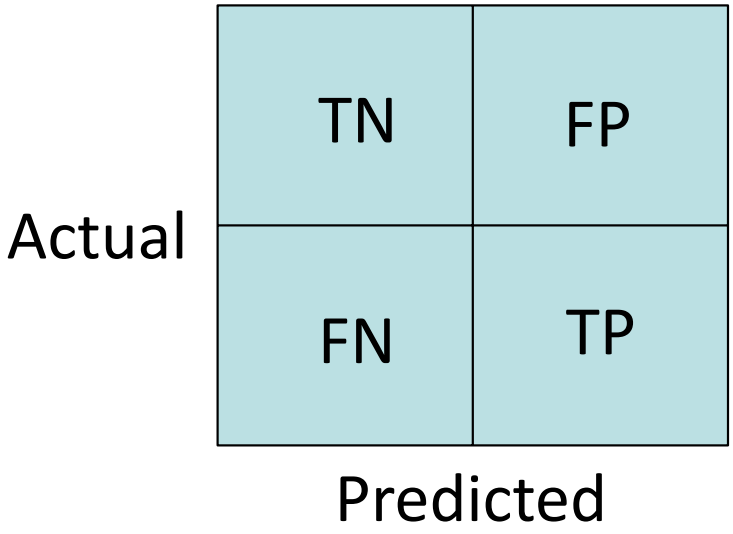
Accuracy

- Definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of all predictions}}$$

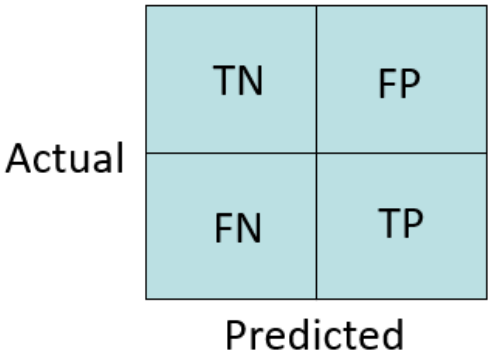
- Accuracy is the simplest and most basic measure

- It is usually not sufficient to assess the performance (see the example on the next slide)



Clarification of the accuracy

- Suppose we have a bunch of images from MNIST
- Our goal is to identify the digit 5
- Each row represents an actual class
- Each column is a predicted class



Non-5 (Class 0)

Digit 5 (Class 1)

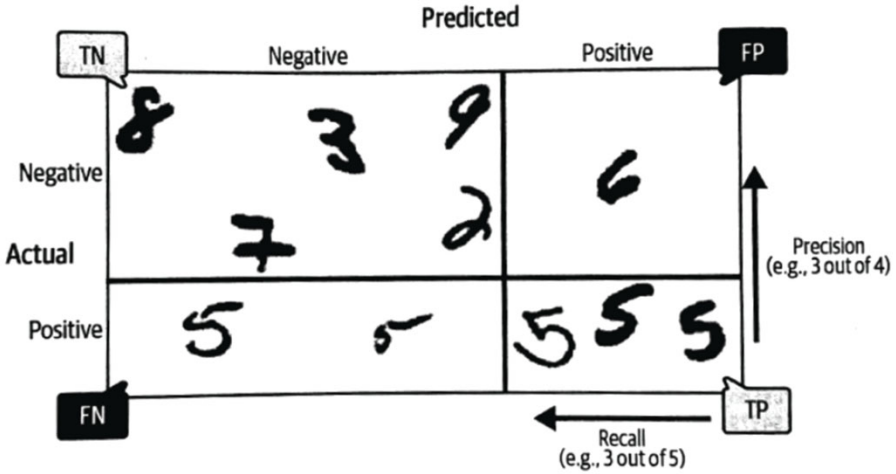


Fig. 3.3 (main reference)

Compare the accuracy measures of two models

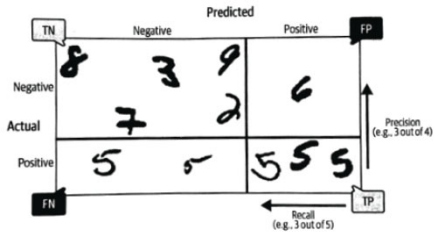
- Suppose you receive 1000 text messages, from which 10 are actual spam
- First model identifies all messages as non-spam
- Second model identifies the actual 10 spam (correct identification), but it also incorrectly identifies another 10 non-spam messages as spam
- Accuracy is usually not the preferred performance measure

Precision and Recall

$$\text{Precision} = \frac{\text{True positive}}{\text{Predicted results}} = \frac{\text{True positive}}{\text{True positive} + \text{ false positive}}$$

$$\text{Recall} = \frac{\text{True positive}}{\text{Actualresults}} = \frac{\text{True positive}}{\text{True positive} + \text{ false negative}}$$

- Other names for Recall are True Positive Rate or Sensitivity
- Two definitions for Confusion Matrix: both are acceptable if you are consistent

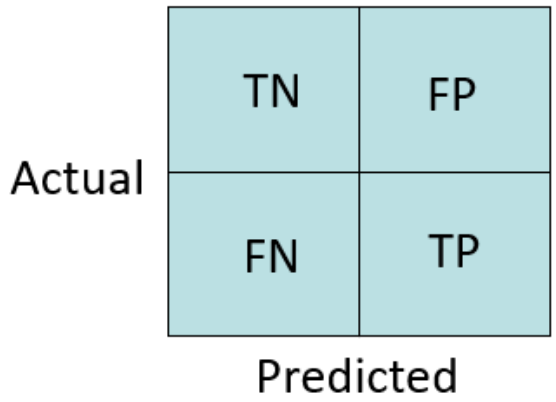


Actual	Predicted	
	TP	FN
	FP	TN

Actual	Predicted	
	TN	FP
	FN	TP

Hypothetical perfect classifier

- It has only true positive and true negative values
- Its confusion matrix is diagonal: non-diagonal terms (FP and FN) are equal to zero



Effects of threshold on the Precision and Recall

- Suppose our goal is to find the digit five
- There are 12 numbers, from which six are 5
- Precision: How correct are the identified numbers (among themselves)?
- Recall: What fraction of the correct answers is collected (from the six samples)?

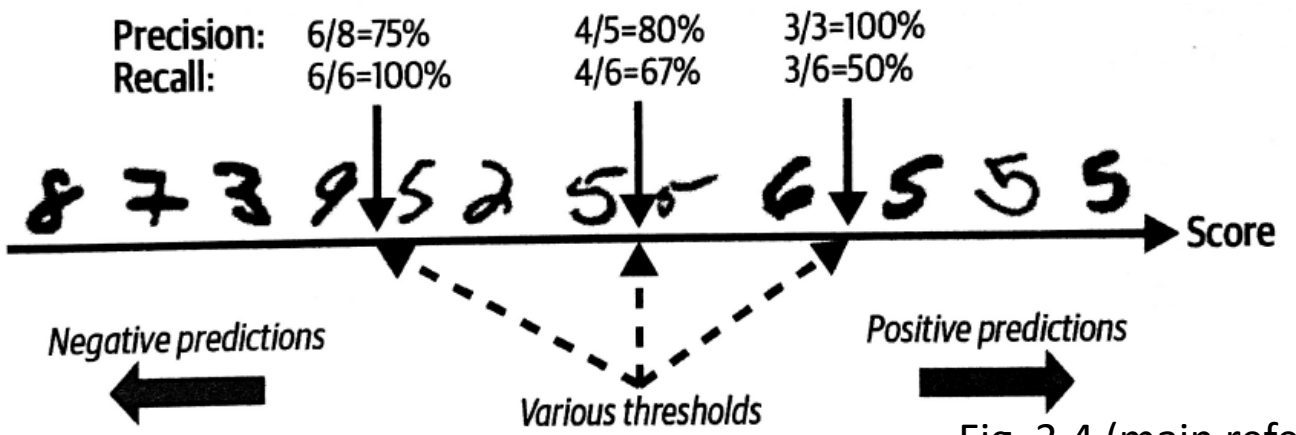


Fig. 3.4 (main reference)

F1 Score (unified measure)

- F1 accounts for the recall and precision

$$\mathrm{F}_1 = \frac{2}{1/\mathrm{precision} + 1/\mathrm{recall}} = 2\,\frac{\mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

- F1 favors classifiers whose recall and precision have similar performance
- In practice, either precision or recall is more important

F_β Score

- This measure gives more weight to Precision or Recall (depending on β)

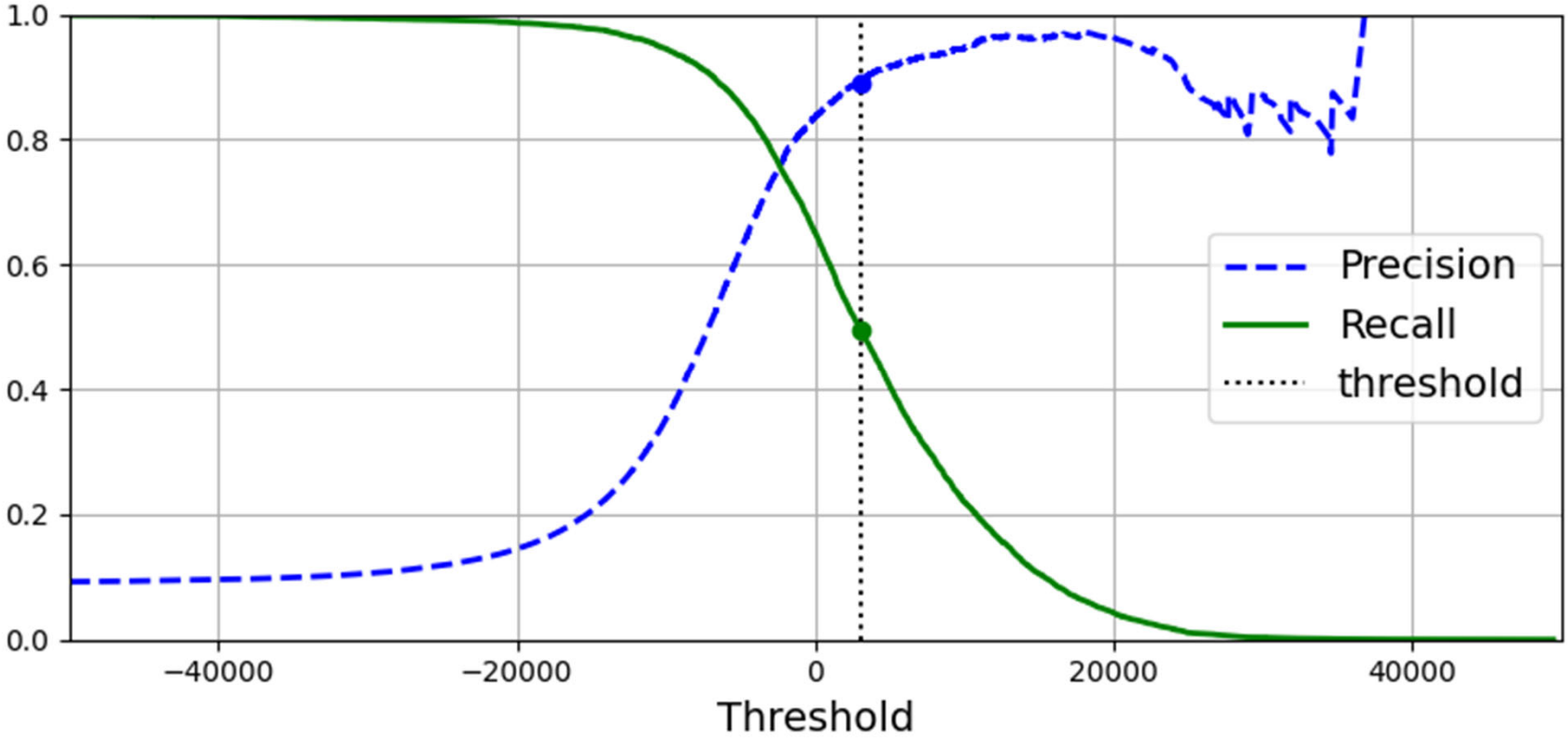
$$F_\beta = (1 + \beta^2) \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

- The parameter (precision or recall) with a lower weight increases more (becomes more important)

Precision and Recall trade-off

- Increasing precision reduces recall
- Increasing recall also reduces precision
- Run the classification code (classification_metrics) posted on CANVAS to determine where the importance of each measure
- This code is a modification of the code available from the main reference

Precision and Recall as a function of the threshold of a classifier

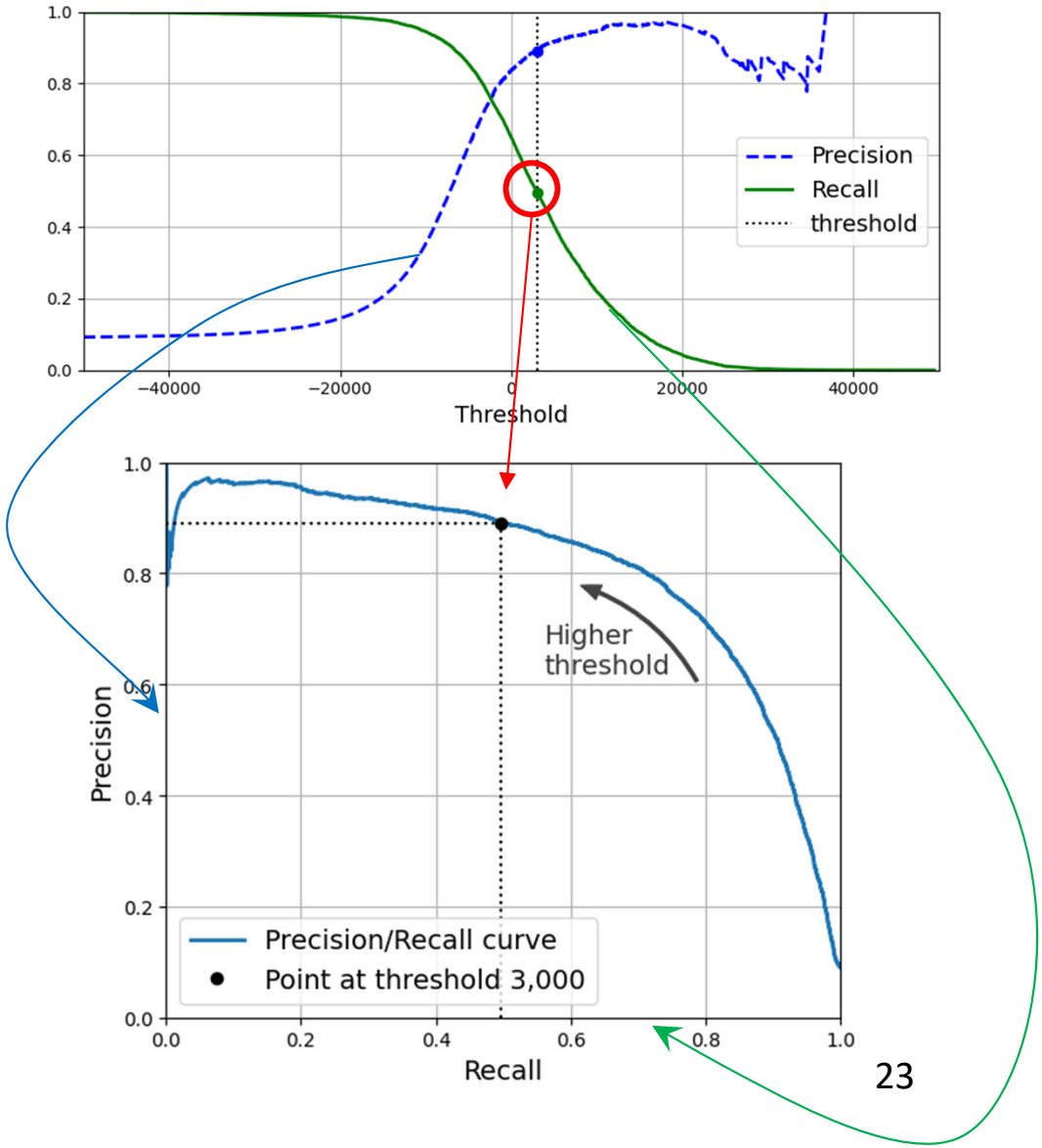


- Q1: Why is the Precision non-monotonic?
- Q2: Which threshold value would you choose? Why?

Precision vs Recall

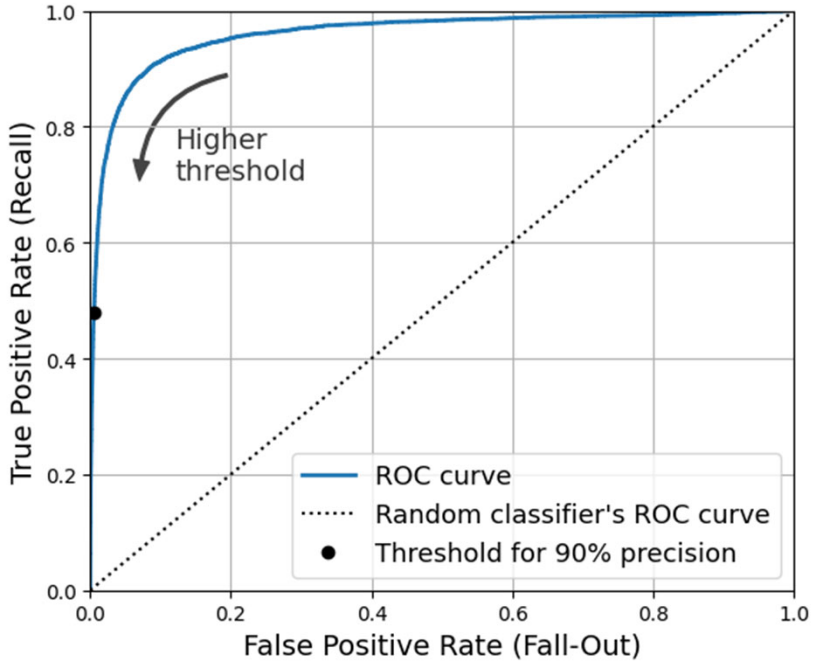
- This is just another presentation of the previous plot

- We can also plot the precision versus recall to assess the model further



Receiver operating characteristic (ROC)

- ROC provides another tool to analyze the model performance
- It shows true positive rate versus false positive rate



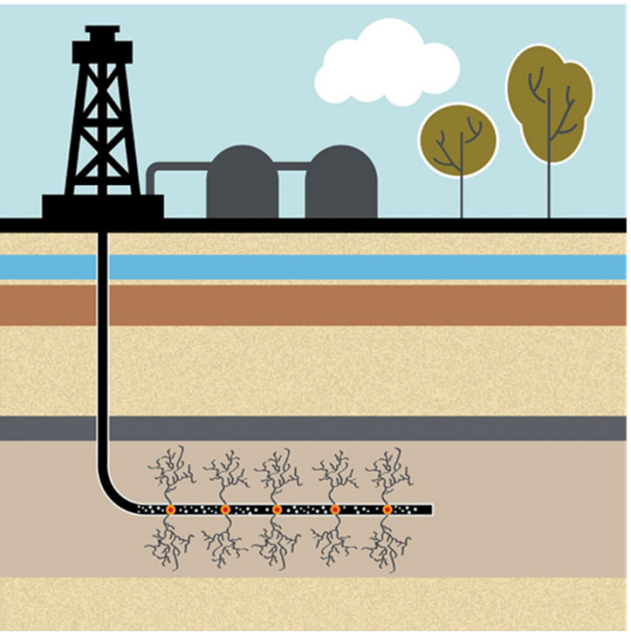
- Q: What was the other name for the true positive rate we defined earlier

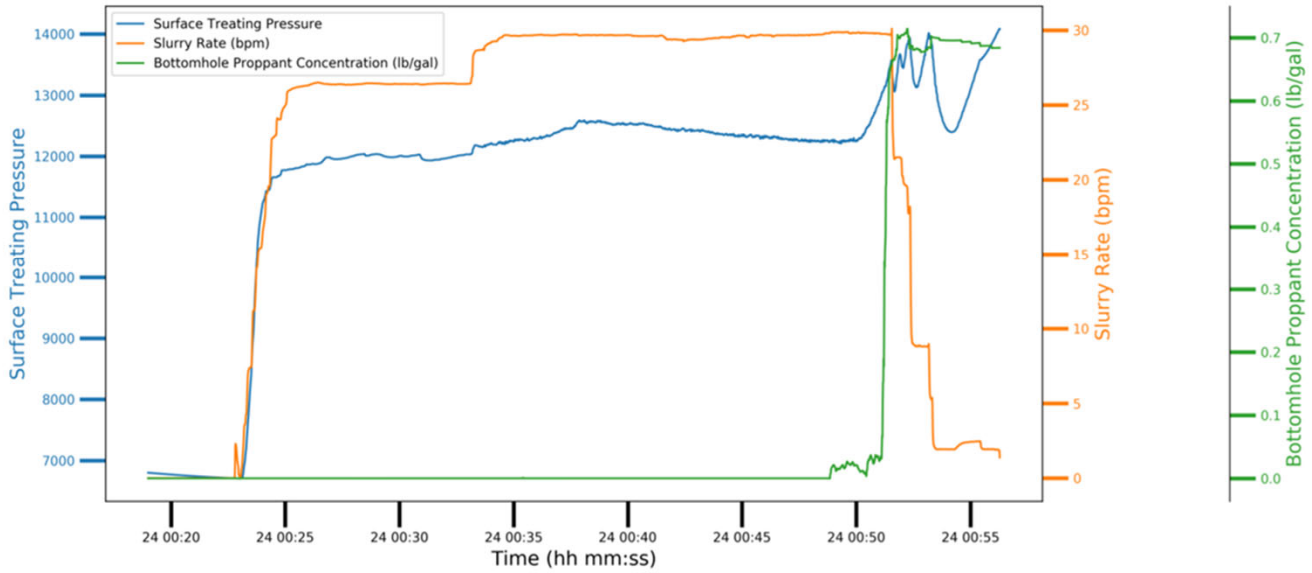
Which plot (PR vs. ROC) would better assess your classifier?

- In general, it would be good to create both to gain a better understanding of the model
- Use the Precision/Recall (PR) curve if the positive class is rare
- Use the PR curve if you care more about the false positive than the false negative
- Otherwise, use ROC

Screen Out Example

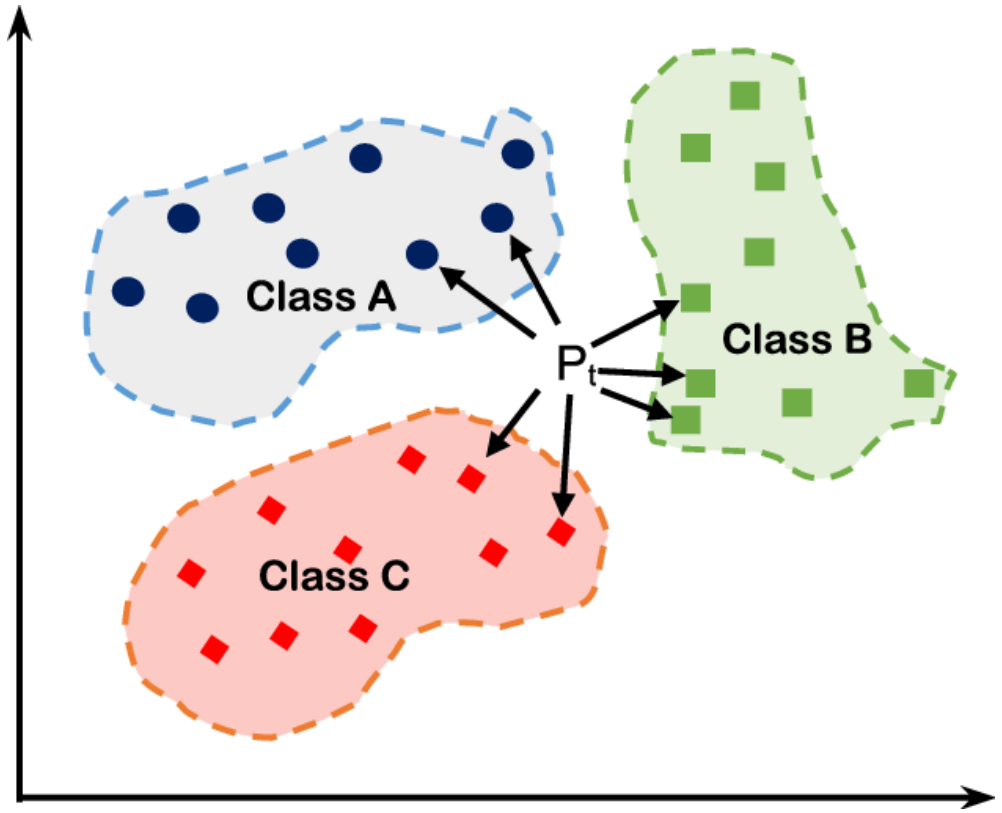
- Classify a screen-out stage ahead of time





high cost associated with false negative, and the goal is to maximize recall

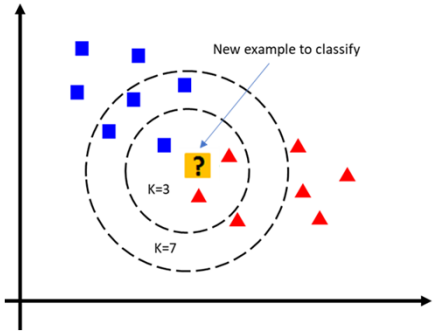
K Nearest Neighbors Classifier (KNN)



Main features of KNN

- It accounts for the closest neighbors' class
- It is non-probabilistic (it does not report the probability of a class-like Regression)
- It does not learn any parameter
- There is no linear combination of parameters
- It is non-linear

Voting system of KNN



- It memorizes (or stores) the training samples and their class
- Because it memorizes, KNN may struggle with unforeseen data
- It uses a majority vote. That is why we usually set K to an odd number
- It can form a complex non-linear boundary

Multiclass classification

- There are two general approaches

- First: Divide your *n* classes into *n* binary classes and apply methods used for binary classification if the classifier does not have a multiclass version (SGDClassifeir, SVC,…)

- Second: Use the multiclass versions of the commands if they are available (LigisticRegression, RandomForest,…)

Metrics for multiclass classification

- Macro averaging: simply average the F1 scores of different classes

$$\text{macro} = \frac{\text{F}_{\text{1class1}} + \text{F}_{\text{1class2}} + \text{F}_{\text{1class3}} + \cdots + \text{F}_{\text{1classN}}}{\text{N}}$$

- Weighting: you may want to weigh the scores based on the number of samples

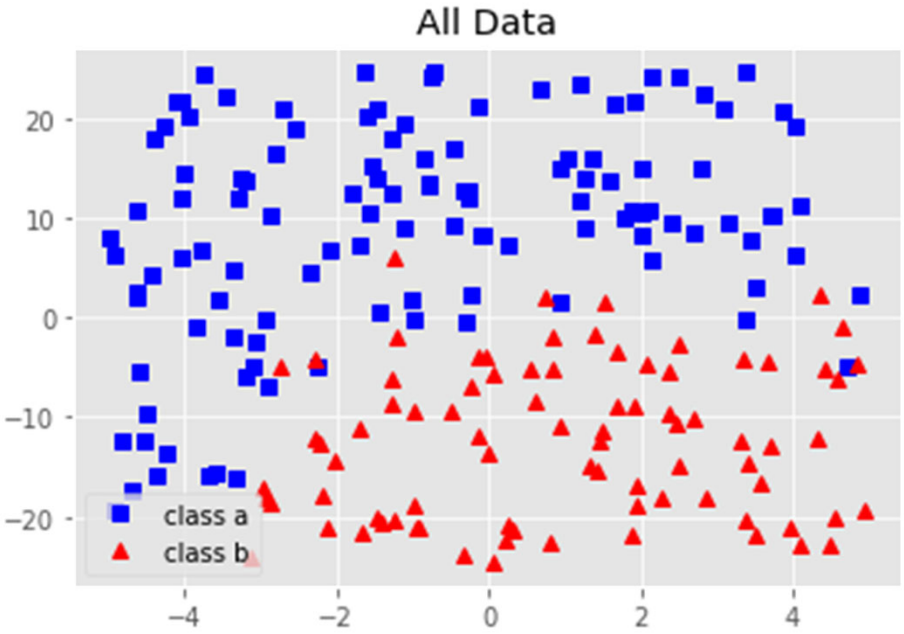
$$\text{weighted} = \text{W}_1 * \text{F}_{\text{1class1}} + \text{W}_2 * \text{F}_{\text{1class2}} + \cdots \text{W}_\text{N} * \text{F}_{\text{1classN}}$$

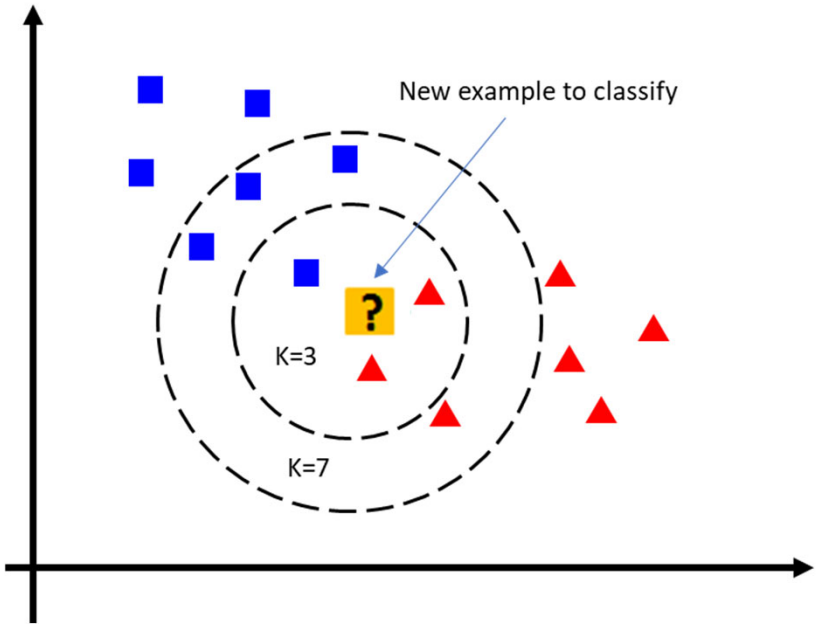
constrain: $\text{W}_1 + \text{W}_2 + \text{W}_3 + \cdots + \text{W}_\text{N} = 1$

KNN code

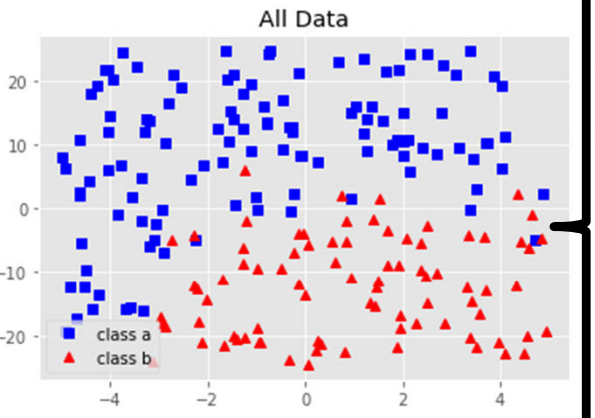
- Run KNN code posted on CANVAS (Classification_KNN_LogisticRegression)

KNN code (continued)



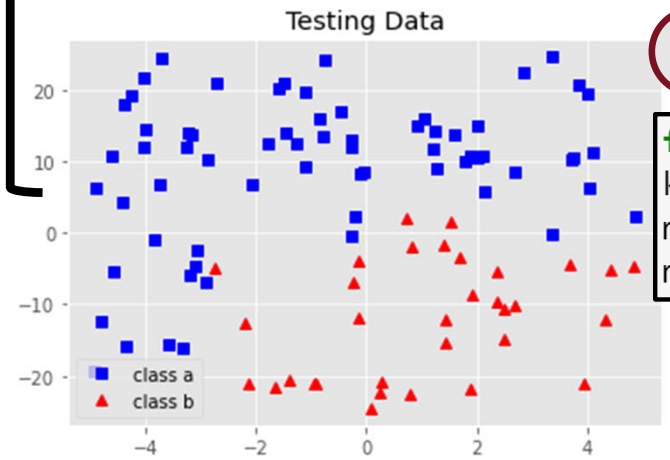


KNN code (continued)



1

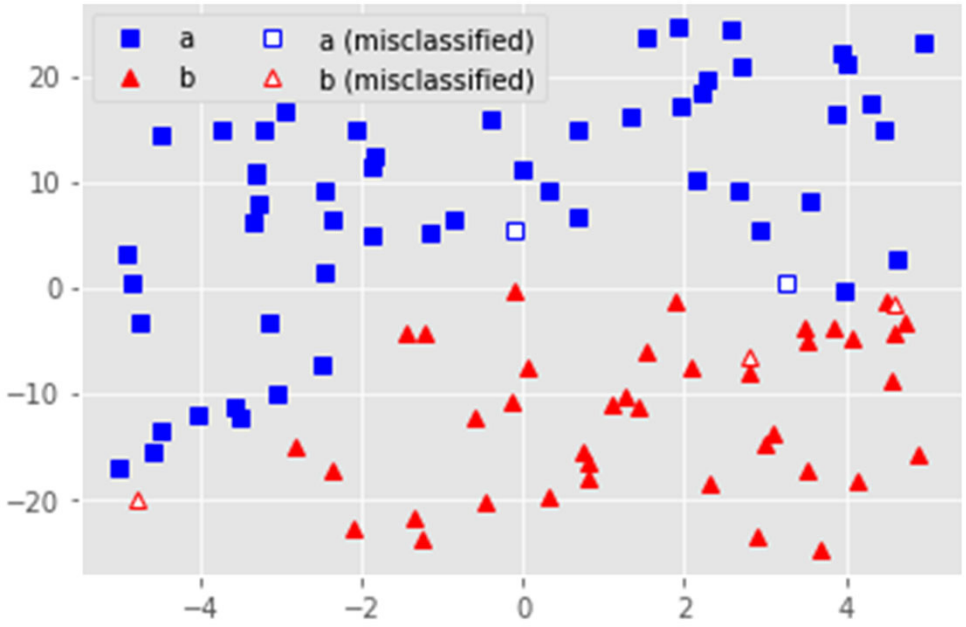
```
train, test = train_test_split(data, test_size = 0.5)
```



2

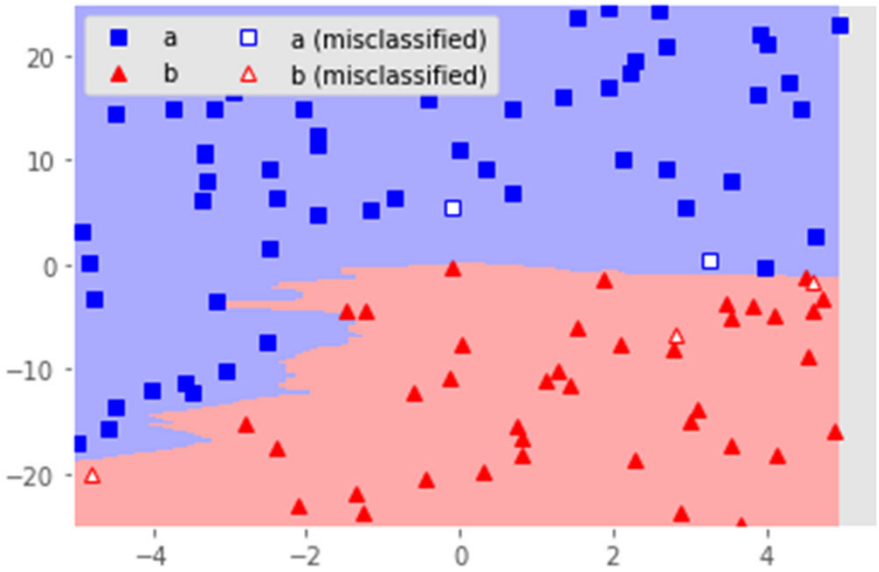
```
from sklearn import neighbors
k = 5
model = neighbors.KNeighborsClassifier(k)
model.fit(train[['x','y']], train['class'])
```

KNN code (continued)




```
def plot_predicted_1(model, test):
    predicted = model.predict(test[['x','y']])
    correct = test[test['class'] == predicted]
    correcta = correct[correct['class'] == 'a']
    correctb = correct[correct['class'] == 'b']
    incorrect = test[test['class'] != predicted]
    incorrecta = incorrect[incorrect['class'] == 'b']
    incorrectb = incorrect[incorrect['class'] == 'a']

    plt.plot(correcta['x'], correcta['y'], 'bs', label='a')
    plt.plot(correctb['x'], correctb['y'], 'r^', label='b')
    plt.plot(incorrecta['x'], incorrecta['y'], 'bs',
             markerfacecolor='w', label='a (misclassified)')
    plt.plot(incorrectb['x'], incorrectb['y'], 'r^',
             markerfacecolor='w', label='b (misclassified)')
    plt.legend(loc='upper left', ncol=2, framealpha=1)

plot_predicted_1(model, test)
```

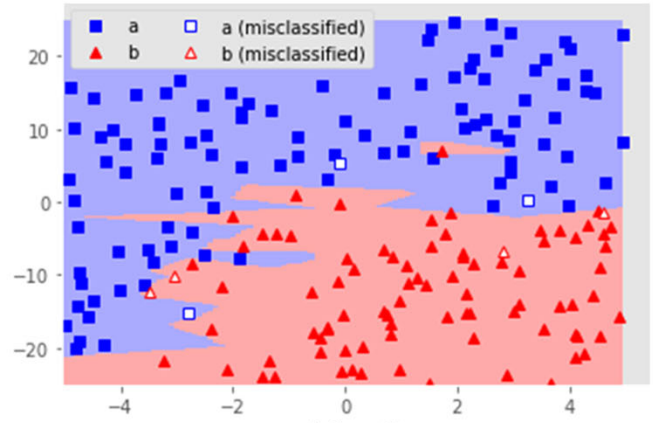
KNN code (continued)



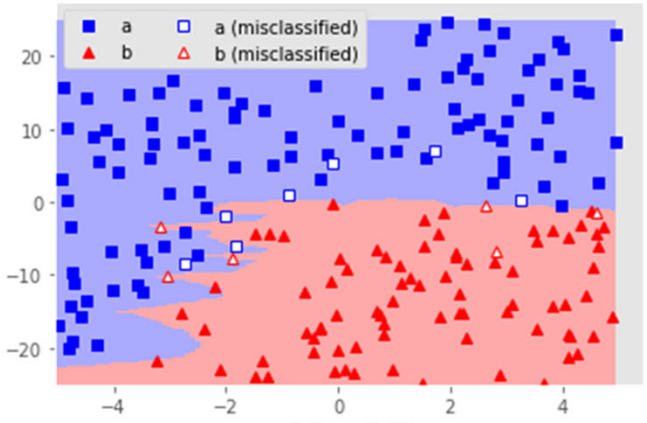

```
def plot_predicted_2(model, test):
    cmap = ListedColormap(['#AAAAFF','#FFAAAA'])
    xmin, xmax, ymin, ymax = -5, 5, -25, 25
    grid_size = 0.1
    xx, yy = np.meshgrid(np.arange(xmin, xmax, grid_size),
                         np.arange(ymin, ymax, grid_size))
    pp = model.predict(np.c_[xx.ravel(), yy.ravel()])
    zz = np.array([{'a':0,'b':1}[ab] for ab in pp])
    zz = zz.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, zz, cmap = cmap)
    plot_predicted_1(model, test)

plot_predicted_2(model, test)
plt.show()
```

Model Complexity with k-Nearest Neighbor

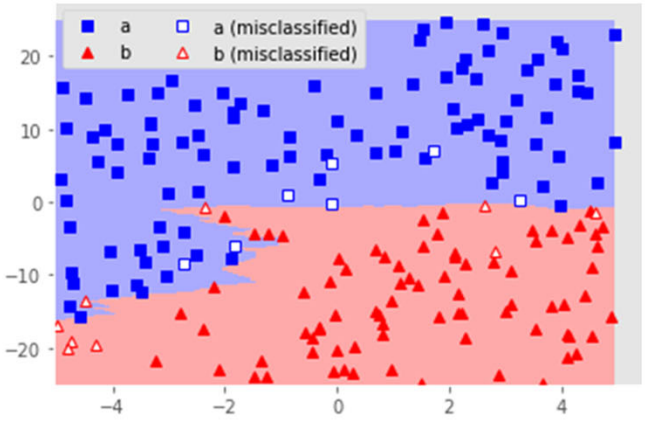
K=1



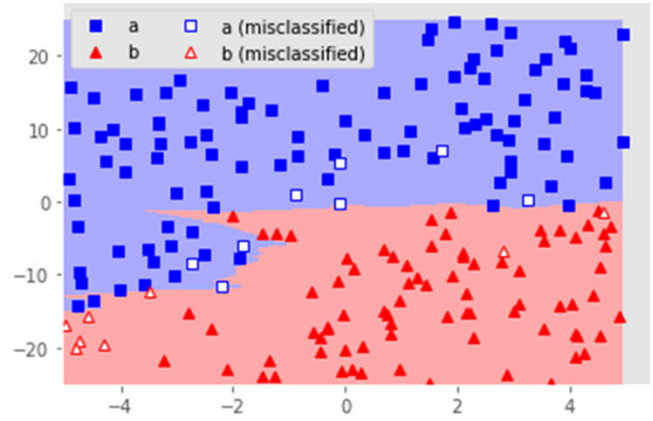
K=3



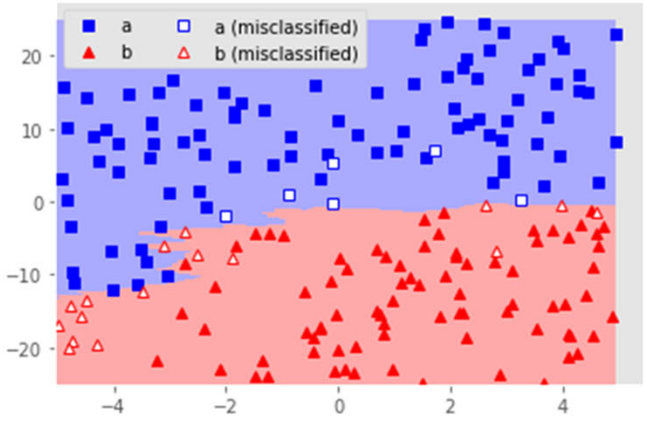
K=7



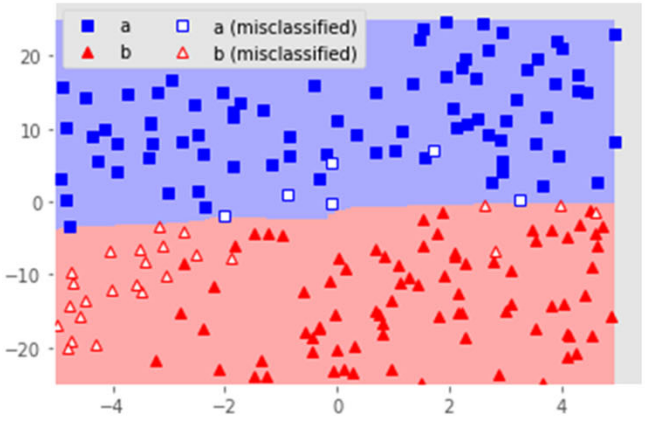
K=9



K=15



K=21



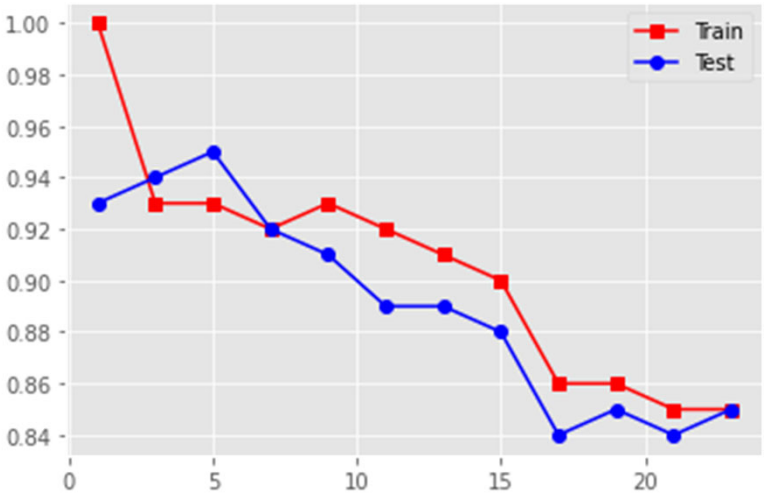
Overfitting vs underfitting in classification

- Similar to the regression, classification is sensitive to the complexity of your model

- In KNN, you overfit by using too few neighbors and underfit by using too many neighbors

Model Score

- The scikit-learn nearest neighbor model can generate a score based on the accuracy of our prediction.




```
kvals = range(1,25,2)
train_score = []
test_score = []

for k in kvals:
    model = neighbors.KNeighborsClassifier(k)
    model.fit(train[['x','y']], train['class'])
    train_score.append(model.score(train[['x','y']], train['class']))
    test_score.append(model.score(test[['x','y']], test['class']))

plt.plot(kvals, train_score, 'r-s', label='Train')
plt.plot(kvals, test_score, 'b-o', label='Test')
plt.legend()
plt.show()
```

Logistic Regression

Logistic Regression

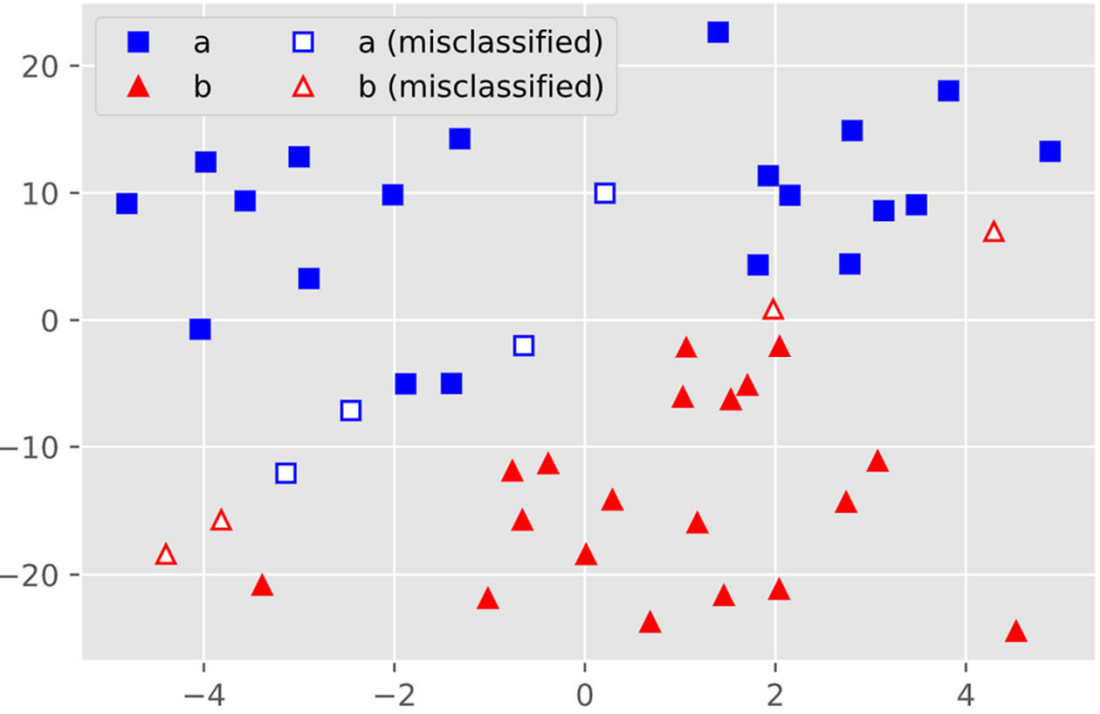
- Logistic regression takes a different approach than KNN
- Rather than modeling the two classes as numbers, it predicts the probability of belonging to a specific group
- The best way to think about logistic regression is that it is a linear regression, but for classification problems
- We fit the model using the logistic function:

$$\text{Logistic function} = \frac{1}{1 + e^{-x}}$$

Common vs acceptable threshold in linear regression

- We often use the probability of 0.5 to map the outcome of logistic function to binary classes (common threshold)
- The threshold can be set equal to 0.7, 0.8, or any other value in the (0,1) range (acceptable threshold)
- The outcome of the sigmoid function does not really become equal to 0 or 1. This means there is always some uncertainty

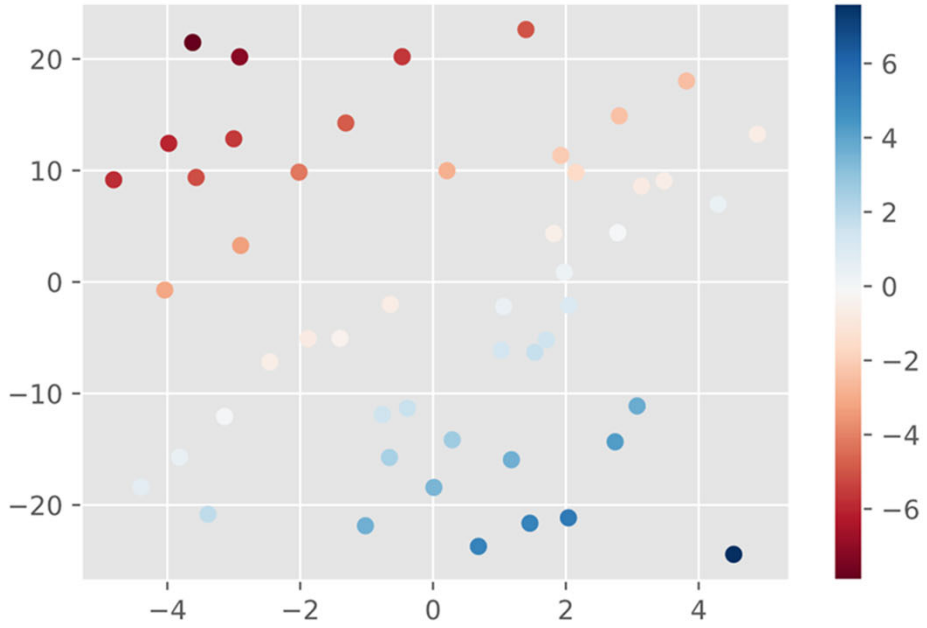
Logistic Regression (results from code)




```
model = linear_model.LogisticRegression()
model.fit(train[['x','y']], train['class'])

def plot_predicted_1(model, test):
    predicted = model.predict(test[['x','y']])
    correct = test[test['class'] == predicted]
    correcta = correct[correct['class'] == 'a']
    correctb = correct[correct['class'] == 'b']
    incorrect = test[test['class'] != predicted]
    incorrecta = incorrect[incorrect['class'] == 'b']
    incorrectb = incorrect[incorrect['class'] == 'a']

    plt.plot(correcta['x'], correcta['y'], 'bs', label='a')
    plt.plot(correctb['x'], correctb['y'], 'r^', label='b')
    plt.plot(incorrecta['x'], incorrecta['y'], 'bs',
             markerfacecolor='w', label='a (misclassified)')
    plt.plot(incorrectb['x'], incorrectb['y'], 'r^',
             markerfacecolor='w', label='b (misclassified)')
    plt.legend(loc='upper left', ncol=2, framealpha=1)

plot_predicted_1(model, test)
```

Decision function of Logistic Regression

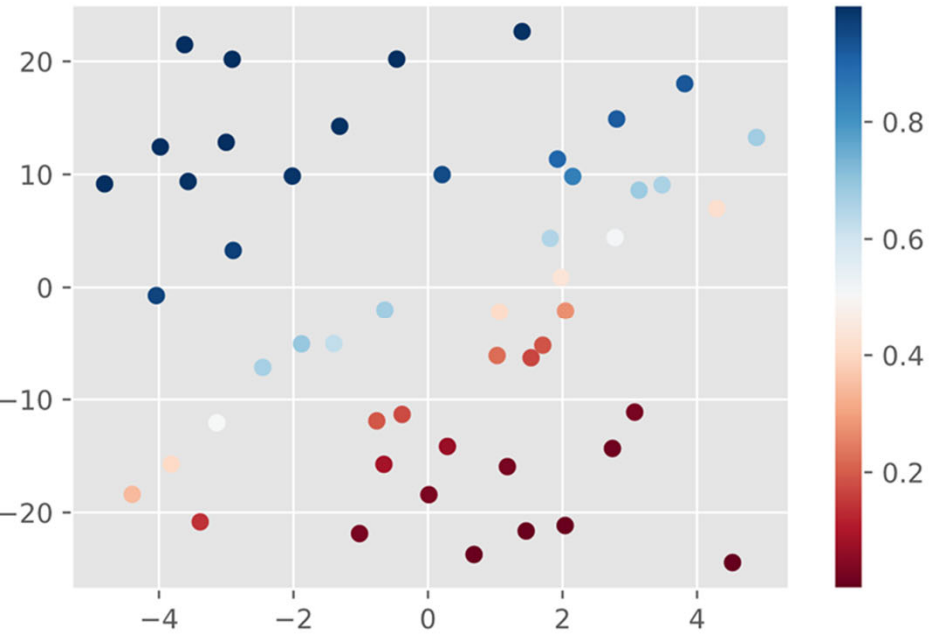
- The logistic regression model also yields a decision function that determines the distance for each point from the dividing hyperplane




```
plt.figure(figsize=(6,4))
plt.scatter(test['x'], test['y'],
            c=model.decision_function(test[['x','y']]),
            cmap='RdBu')
plt.colorbar()
plt.show()
```

Generated probabilities of Logistic Regression

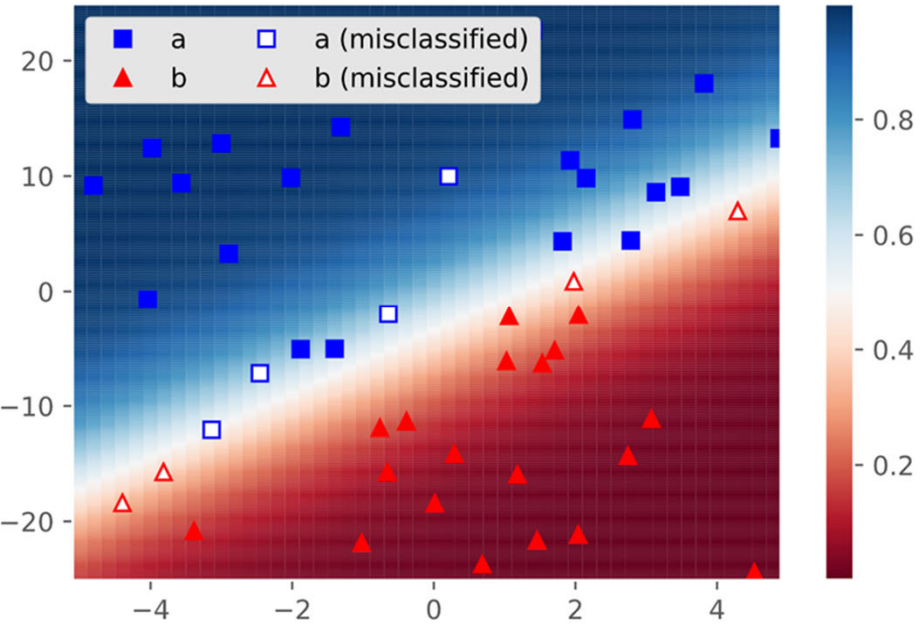
- We were estimating class membership probabilities
- This means we can retrieve these probabilities




```
plt.figure(figsize=(6,4))
plt.scatter(test['x'], test['y'],
            c=model.predict_proba(test[['x','y']])[:,0],
            cmap='RdBu')
plt.colorbar()
plt.show()
```

Separating plane of Logistic Regression

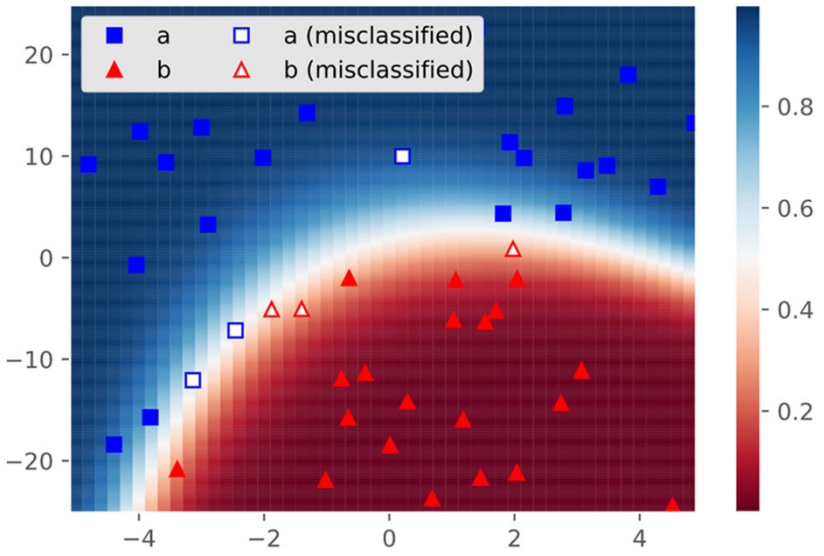
- We can also plot the probabilities in the plane




```
def plot_probabilities(model, test):
    cmap = 'RdBu'
    xmin, xmax, ymin, ymax = -5, 5, -25, 25
    grid_size = 0.2
    xx, yy = np.meshgrid(np.arange(xmin, xmax, grid_size),
                         np.arange(ymin, ymax, grid_size))
    pp = model.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:,0]
    zz = pp.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, zz, cmap = cmap)
    plt.colorbar()
    plot_predicted_1(model, test)

plot_probabilities(model, test)
plt.show()
```

Logistic Regression with 2nd order polynomial

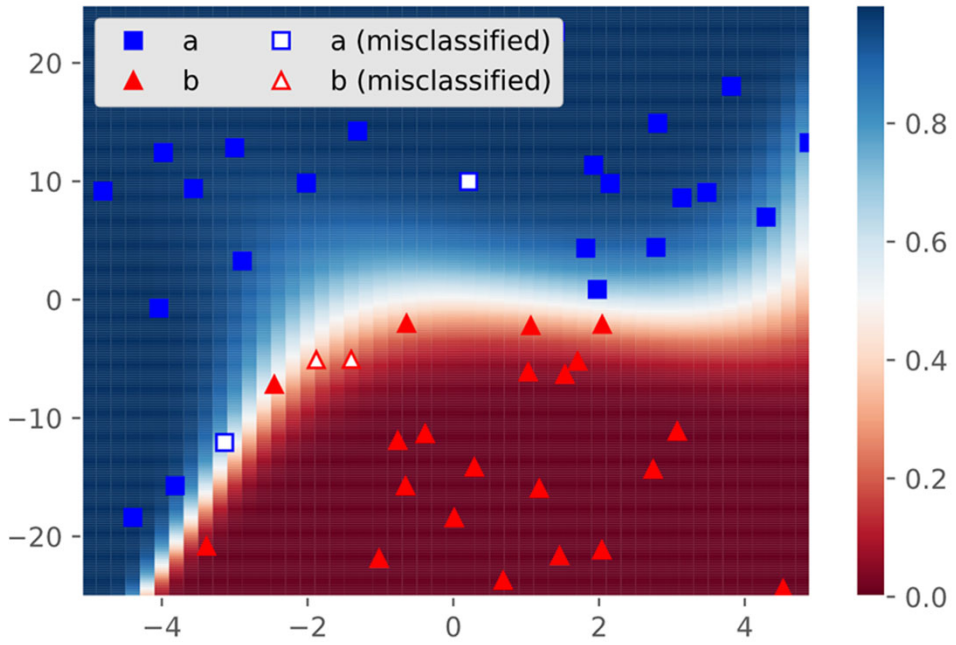
- Simple polynomial model:
  - Using sklearn.preprocessing.PolynomialFeatures We can generate polynomial expansions of our base features to any degree desired
  - New features $1, x, y, x^2, xy, y^2$



Test score: 0.88

To-Do: Extend the code yourself

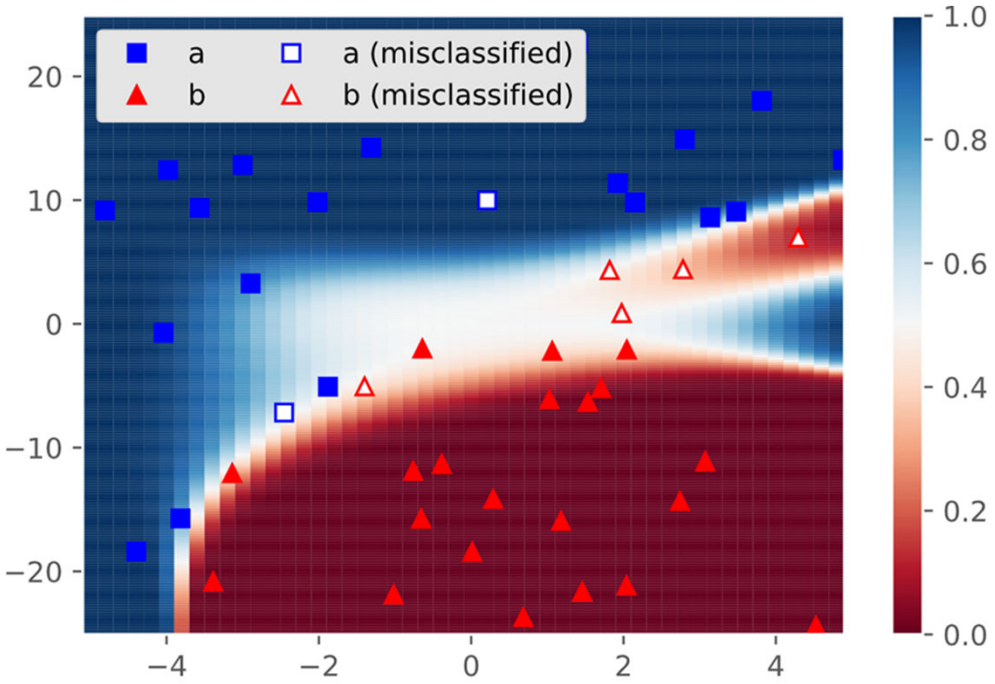
Logistic Regression with 3rd order polynomial



Test score: 0.92

To-Do: Extend the code yourself

Logistic Regression with 4th order polynomial

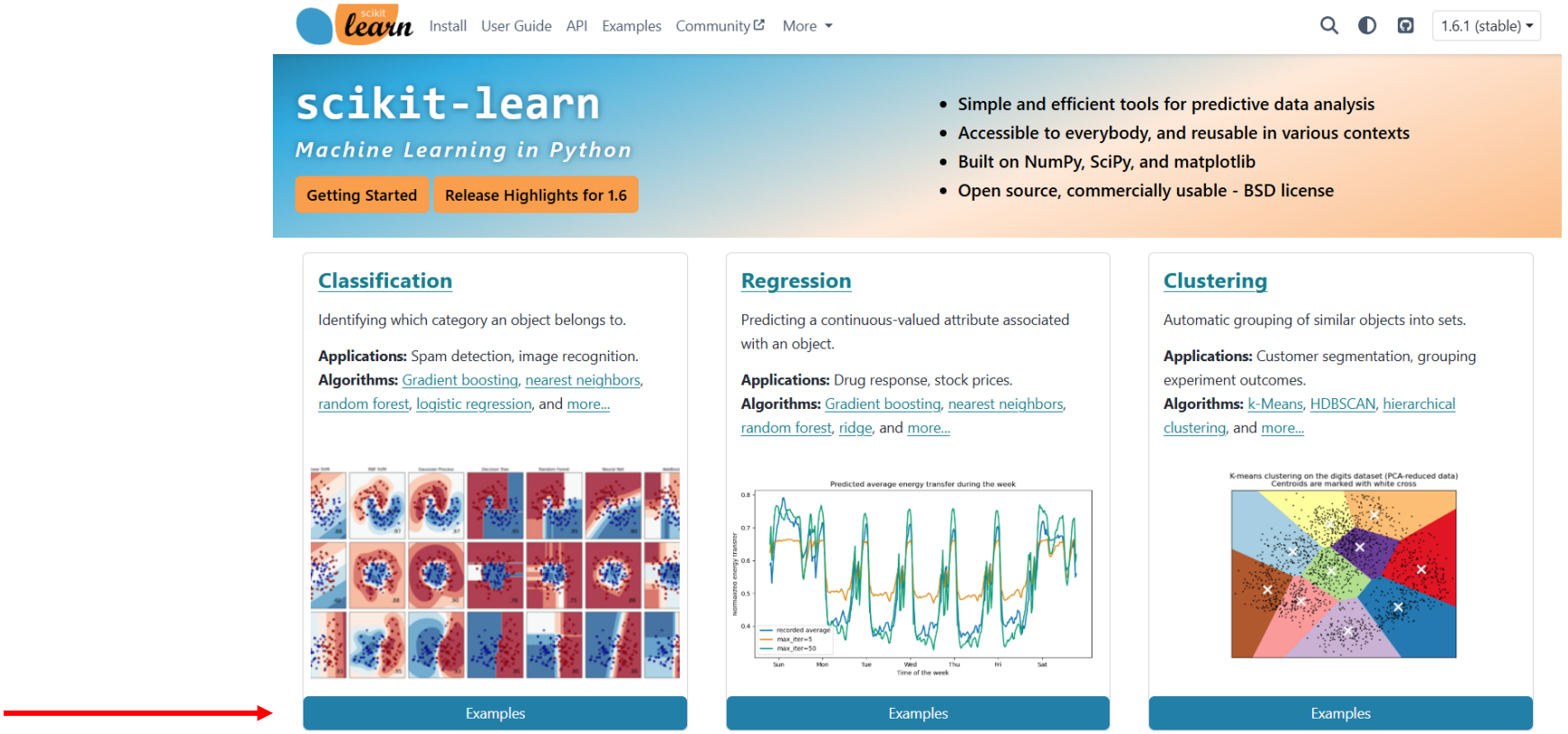


Test score: 0.86

How can we run another classifier that is not discussed in class?

Check sample codes on scikit-learn

- Go to scikit-learn main page (https://scikit-learn.org/stable/index.html)

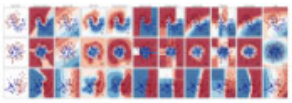


Go to the classifier comparison page

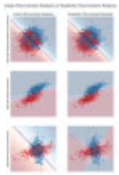
- Download the code accessible from the link below

Classification

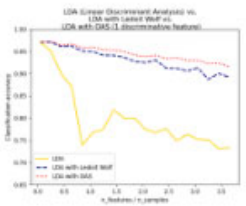
General examples about classification algorithms.



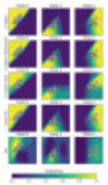
Classifier comparison



Linear and Quadratic Discriminant Analysis with covariance ellipsoid



Normal, Ledoit-Wolf and OAS Linear Discriminant Analysis for classification



Plot classification probability

Recognizing hand-written digits

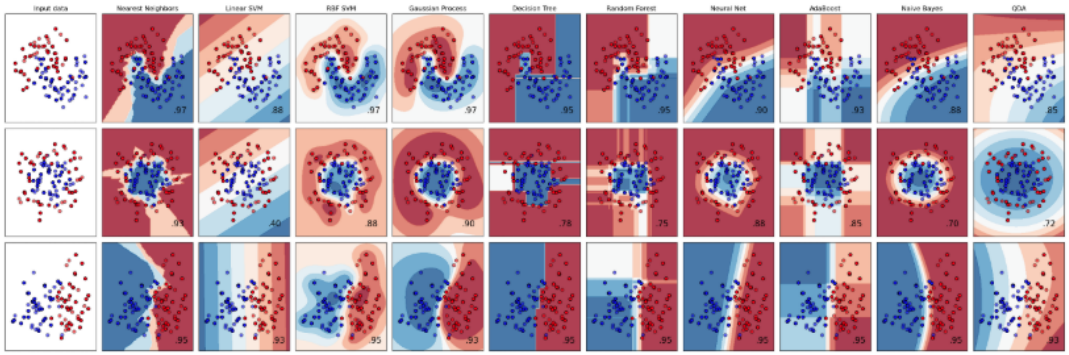
Run the downloaded code on your machine to compare them

Classifier comparison

A comparison of several classifiers in scikit-learn on synthetic datasets. The point of this example is to illustrate the nature of decision boundaries of different classifiers. This should be taken with a grain of salt, as the intuition conveyed by these examples does not necessarily carry over to real datasets.

Particularly in high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as naive Bayes and linear SVMs might lead to better generalization than is achieved by other classifiers.

The plots show training points in solid colors and testing points semi-transparent. The lower right shows the classification accuracy on the test set.

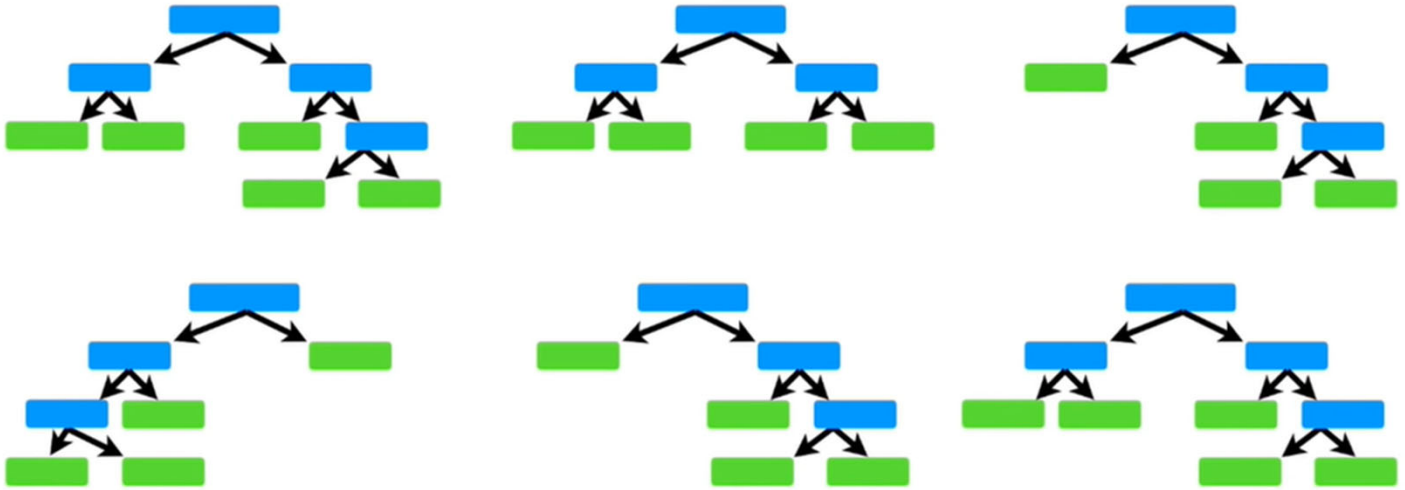


- Q: How would you apply different classifiers to your data?

Random Forest

Random Forest Classification are Ensemble Machine Learning

- Weak Learners form Strong Learns with lower variance and better performance
- Reduces variance with minimal increase in bias
- Difficult to interpret and lack of transparency as compared to decision tree (discussed later in the course)



Access the code on CANVAS

- Code name: RandomForest
- Also upload the data file: TOC_Prediction_DataSet
- Discuss the results