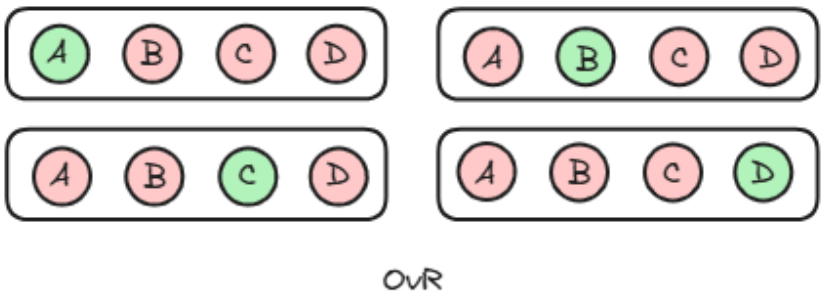
06.09. 분류

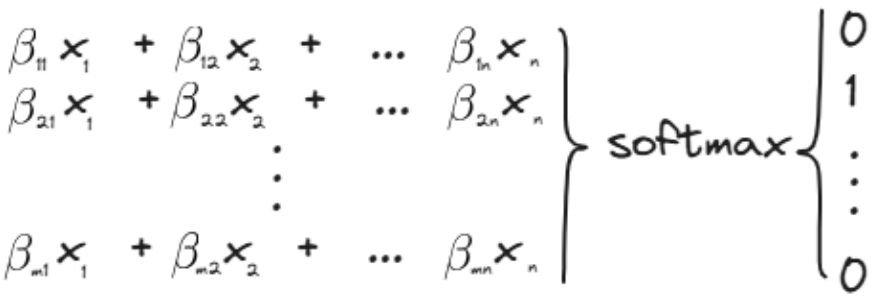
1. Logistic Regression의 multi-class 분류

- sigmoid를 사용해 두 개의 라벨은 분류 가능. 근데 3개, 4개는...?

1. OvR(One vs Rest) 방식 : 하나의 라벨과 그 외의 라벨들로 분류하는 n개의 분류기를 만들어 결과를 취합한다.



2. Multinomial 방식 : softmax 함수를 사용해 여러 개의 라벨로 한 번에 분류한다.



- softmax : $\frac{e^x}{\sum e^x}$

→ scikit-learn의 경우 OvR 방식이 기본이지만 앞으로는 Multinomial 방식을 기본으로 사용하게 됨!

LogisticRegression의 파라미터

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

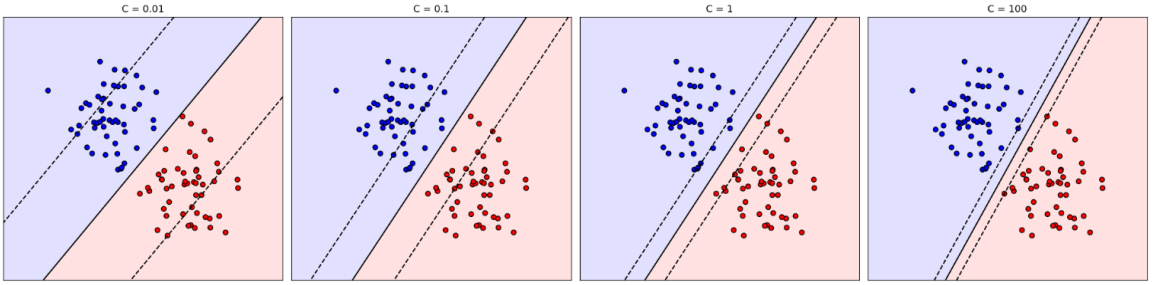
- **solver** : ***{'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}***
  - linear regression과 달리 해를 바로 구할 수 없기 때문에 최적화 과정을 거침. 위 solver들은 최적화 과정을 어떻게 수행할 것인지와 관련된 파라미터들임.
  - **newton-cg / newton-cholesky** : 정밀한 결과를 얻고 싶을 때 사용. (l2)
  - **lbfgs** (기본값) : 대부분의 상황에 적합. (l2)
  - **liblinear** : 작고 단순한 문제에 적합. (l1, l2)
  - **sag, saga** : 데이터가 많을 때 적합. (속도 측면에서) (l1, l2(saga만))
- **penalty** : *'l1', 'l2(기본값)', 'elasticnet'*
  - LinearRegression의 규제 버전인 'Lasso', 'Ridge', 'ElasticNet'과 같은 기능.
  - **l1** : 해석이 쉬운 모델이 필요하거나 일부 특성만 필요할 때 사용
  - **l2** : 모든 변수를 사용하는 것이 유리하거나 변수 간의 상관관계가 높을 때 유용
  - **elasticnet** : l1 정규화와 l2 정규화를 모두 활용. (solver가 saga일 때만 사용 가능) (l1_ratio인자를 추가로 줘야 함.)
- **C** : *기본값 (1). (0이상의 실수)*
  - 작을수록 규제가 강해진다.
- **class_weight** : *'None(기본값)', 'balanced'*

- 라벨 불균형이 있을 경우 'balanced'를 사용하면 라벨이 적은 데이터에 더 큰 가중치를 준다.

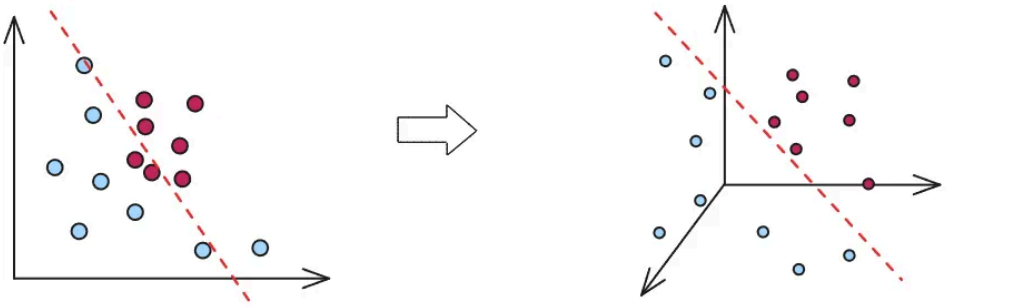
2. SVM 활용

SVM의 Hard Margin 과 Soft Margin

- SVC의 C 파라미터로 이를 결정. (커질수록 Hard margin에 가까워짐.)



선형 함수로 경계를 나누기 애매한 경우

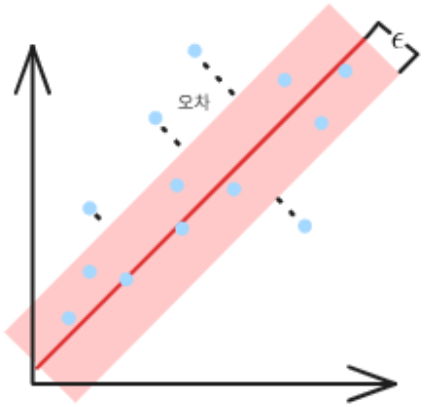


선형적으로 분리되기 어려움

고차원 공간으로 매핑하면 위와 같이 선형으로 분리될 수 있다.

- 커널 트릭을 활용한 고차원 매핑을 통해 경계를 구분한다.

SVM으로 어떻게 회귀를 수행할까?



- 선을 그었을 때, 선을 기준으로 위아래로 ϵ만큼의 간격 밖에 있는 데이터와의 오차를 계산.
- 해당 오차가 최소화 되는 직선을 찾는다. (= 대부분의 점을 영역 안에 넣는다.)

멀티 클래스 분류

- 역시 OvR 방식으로 해결.

SVC의 파라미터

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

- C : 규제 정도. 클수록 Hard margin에 가까워진다.
- kernel : (***'linear', 'poly', 'rbf', 'sigmoid'***)
  - 커널 트릭에 사용되는 커널.

3. 나이브 베이즈 분류기

- 나이브 베이즈 분류기가 텍스트에 잘 통하는 이유?
  - 중요한 단어들만 잘 잡아내면 됨.
  - 계산이 빠르고 가벼움.
  - 단순한 모델이라 과적합 위험도 적음.

4. KNN

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

KNeighborsClassifier(n_neighbors)

- n_neighbors : K개의 이웃을 선정.
- *distance* : 투표 때 가중치를 설정.
    - ‘uniform’ : 모든 거리에 동일한 가중치.
    - ‘distance’ : 가까울수록 더 큰 가중치.
- *algorithm* : K개의 이웃을 찾는 방법.
    - auto : 자동으로 제일 빠른 방법을 선택.
    - ball_tree : 고차원 데이터에 적합.
    - kd_tree : 저차원 수치형 데이터에 적합.

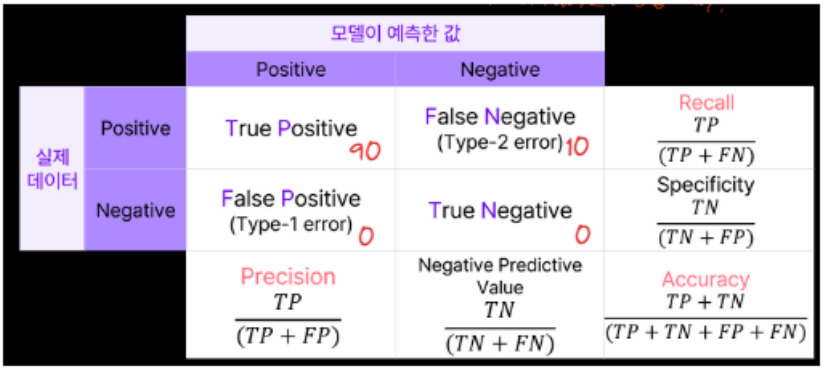
5. 어떤 분류 모델을 써야 할까?

- 데이터가 텍스트 형태 → Naive Bayes
- 데이터가 작고, 형태를 잘 모르겠다 → KNN
- 단순한 데이터 → Logistic Regression
- 복잡한 데이터 → SVM
    - 데이터가 단순한 지 복잡한 지 아는 법?
        - 입력 특징의 수가 적거나, 시각화 했을 때 비교적 구분이 쉬운 경우.
        - 그래도 모르겠으면 Logistic Regression으로 해보고 성능이 만족스럽지 않을 때 SVM을 써보면 된다!
- 결국은 써봐야 안다.

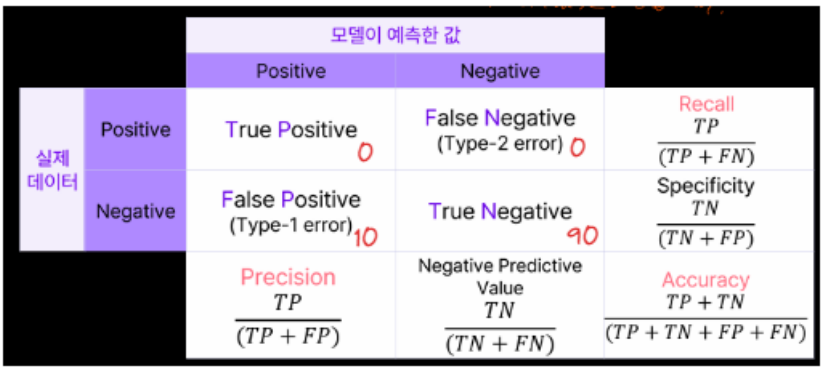
6. 모델 평가 지표

Positive와 Negative

- 비행기 지연 여부 판별하는 모델에서, 지연 데이터가 90개, 비지연 데이터가 10개라고 가정해보자.
- 이 때, 모델이 모든 데이터를 지연으로 판별한다고 하면 정확도는 90%가 된다.
- 그렇다면 Precision이나 Recall은 어떻게 될까?
    - positive = 지연



    - Precision = 1 / Recall = 0.9
    - 여전히 높은 것을 볼 수 있다.
  - positive = 비지연



    - Precision = 0 / Recall = 0
    - 완전히 망해버린 것을 알 수 있다.
- 즉, Recall / Precision / F1 score는 양성을 뭐로 보냐에 영향을 크게 받는 지표들!
- 어떤 클래스가 더 중요한 지를 먼저 정의해서 positive로 놓고 객관적인 지표를 계산해야 한다!

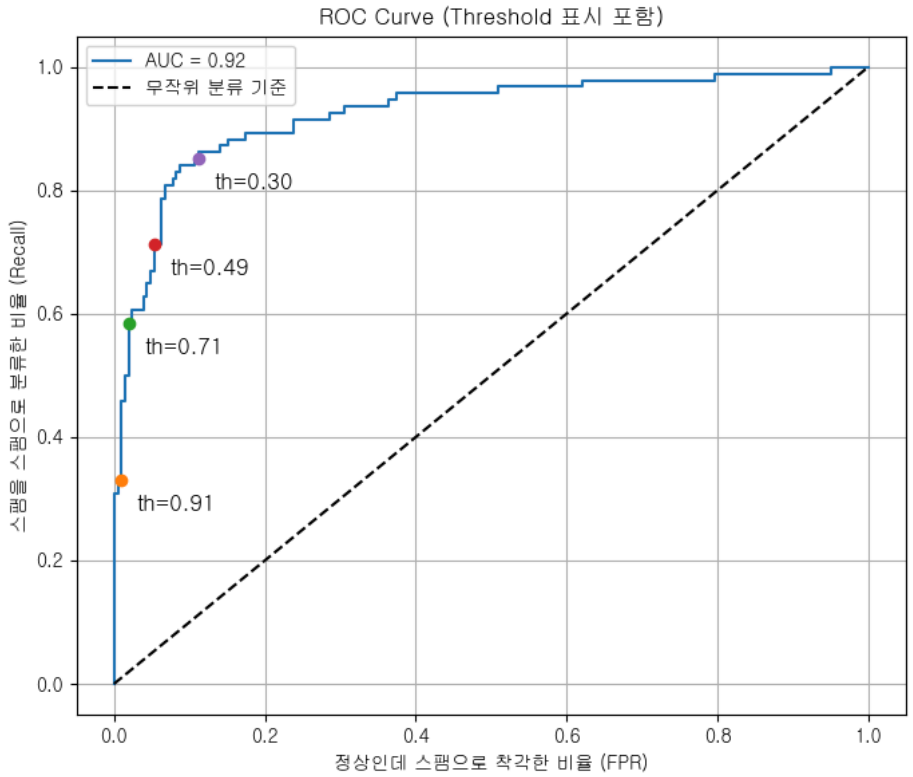
다중 분류에서 F1 Score

- 다중 클래스 분류 문제에서 F1 score를 계산하는 2가지 방법
  - micro : 모든 라벨의 TP/FP/FN을 전부 합쳐서 f1 score 계산.
  - macro : 각 클래스 별 f1 score를 평균.

	Macro	Micro
라벨 불균형	클래스 불균형에 민감	클래스 불균형에 강함

사용	소수의 클래스가 중요할 때	전체적인 정확도가 중요할 때

ROC-Curve 해석

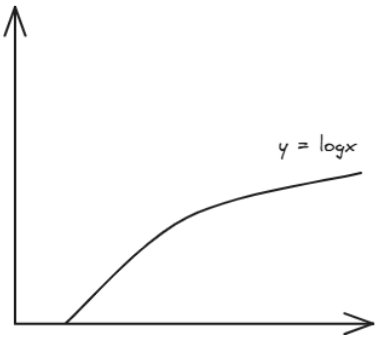


- Threshold 변경에 따른 Recall 과 FPR 수치의 변화를 그린 그래프
  - Threshold란 : 모델이 0~1 사이 값을 예측 했을 때, 어느 비율을 넘어야 스팸으로 분류할 지 기준을 정하는 것.

    ex) threshold = 0.6 → 예측 값이 0.6 이상이면 스팸으로 분류.
  - 즉, threshold가 높아질수록 스팸을 스팸으로 분류할 확률이 낮아진다.'
- 그래프의 왼쪽 위 : 스팸을 스팸으로 분류하면서도, 정상을 스팸으로 분류하지 않는다는 의미.
- 그래프가 왼쪽 위에 가까워 진다 : threshold가 높아져도 스팸을 스팸으로 분류하면서 정상을 스팸으로 분류하지 않고 있다는 의미.
- 그래프의 면적 (AUC)
  - 크다 (1에 가깝다) = 그래프가 왼쪽 위에 가깝게 그려지고 있다 = 성능이 뛰어나다.
  - 작다 (0.5에 가깝다) = 그래프가 왼쪽 위에서 멀게 그려지고 있다 = 성능이 별로다

정규화

- MinMaxScaler()
  - 이상치에 매우 민감. → 이상치가 0 또는 1에 가깝게 몰릴 수 있음.
  - 극단치가 없을 때 사용하면 좋음.
- StandardScaler()
  - 이상치에 덜 민감. → 이상치의 **상대적 위치를 보존**한다.
  - 데이터가 정규 분포를 따를 경우 좋음.
- RobustScaler()
  - 이상치가 많을 때 사용.
- np.log1p
  - 값의 대부분이 0에 몰려있고, 값의 스케일이 매우 클 때 사용하면 좋음.



log 그래프는 위로 갈수록 경사가 완만해지는 형태를 가짐.

- 주의 : 위 기준들은 이론적인 기반으로 정확한 결과는 모두 실험해 보는 것이 좋음!!
  - 이상치가 꼭 성능 저하를 야기하는 것은 아니다.
  - 모델 종류에 따라도 결과가 달라질 수 있다.
  - 머신러닝은 여러가지 요인들이 영향을 주기 때문에 직접 실험해 보는 것이 중요!