Big Contest Futures League

대학교	학과	이름
순천대학교	정보통신공학	배영환

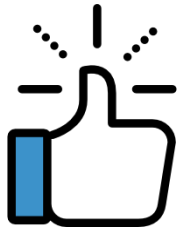
진행 과정

데이터 파악 및 선정

핵심 요점 파악 및
데이터 활용 방안,
데이터 활용 범위 선정

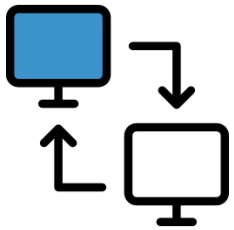
전처리

분석 모델에 맞춰 데이터 형식을 변경
기본적으로 Pandas를 사용
LabelEncoder, One-hot-encoding 사용



최적의 모델 선정

머신러닝 기반의 Sklearn 을 이용한
데이터 분석



모델 적용

최적의 모델을 데이터에 적용

데이터 파악 및 선정

AFSNT	AFSNT	AFSNT_DLY
비행기 및 항공사	비행기 및 항공사	비행기 및 항공사
날짜	날짜	날짜
지연 및 결항	비행기별 운행 요일	지연 여부 및 예측

데이터 파악 및 선정

날씨관련 지연

- 날씨관련 지연은 사용하지 않음 (A로 시작하는 지연)
- 한달 후의 날씨 예측에 대한 불확실
- 비행기에 의한 지연에 비해 비중에 비해 매우 적음

연/월/일

- 연 / 월 / 일 같은 반복되는 특성의 상관이 매우 떨어진다고 판단
- 연 / 월 / 일 대신 요일만 사용하기로 결정

비행기별 날짜 및 계획시간

- 비행기에 속해 있는 요일 및 계획시간에 대한 데이터를 사용하지 않기로 결정
- 모든 데이터의 속성은 AFSNT의 편명에 전부 들어가 있다고 판단

계획 시간

- 계획 시간의 경우 날씨에 관련된 지연여부의 중요도가 높다고 판단
- 날씨에 대한 분석은 하지 않을 것이기 때문에 사용하지 않기로 결정

데이터 파악 및 선정

AFSNT	AFSNT_DLY
비행기 및 항공사 (ARP, ODP, FLO, FLT)	AFSNT 의 Columns
요일 (SDT_DY)	지연 여부 (DLY)
이착륙 (AOD)	지연 예측 (DLY_RATE)

전처리

필요한 데이터만 남겨 놓기 위한 전처리

```
In [8]: Test = AFSNT.copy()
        Test.drop(["SDT_YY", "SDT_DD", "SDT_MM", "REG", "IRR", "STT", "ATT"], axis = 1, inplace = True)
        b =((Test["CNR"] != 'AO5') & (Test["CNR"] != 'AO1') & (Test["CNR"] != 'AO2') &(Test["CNR"] != 'AO7') & (Test["CNR"] != 'AO3
        a = ((Test["DRR"] != 'AO5') & (Test["DRR"] != 'AO1') & (Test["DRR"] != 'AO2') & (Test["DRR"] != 'AO7') & (Test["DRR"] != 'A
        Test["DLYCNL"] = 0
        Test.loc[((Test["DLY"] == "Y") & a) | ((Test["CNL"] == "Y") & b) , ["DLYCNL"] ] = 1
        Test.drop(["DRR", "DLY", "CNL", "CNR"], axis=1, inplace=True)

In [9]: Test.head()
```

Out[9]:

	SDT_DY	ARP	ODP	FLO	FLT	AOD	DLYCNL
0	일	ARP1	ARP3	A	A1901	D	0
1	일	ARP1	ARP3	A	A1905	D	0
2	일	ARP1	ARP3	L	L1751	D	0
3	일	ARP1	ARP3	F	F1201	D	0
4	일	ARP3	ARP1	A	A1900	D	0

필요 없는 데이터를 날리고 필요 있는 데이터만 선별 하는 과정

Pandas 를 주축으로

Boolean indexing 및 map 함수를 이용하여 전처리를 진행

분석 모델 형식에 맞게 전처리

```
In [10]: le = LabelEncoder()
         X = Test.iloc[:, :].values
         X[:, 4] = le.fit_transform(X[:, 4])
         Test["FLT"] = X[:, 4]
         Test1 = Test.drop("FLT", axis =1)
         Test_dummie = pd.get_dummies(Test1.iloc[:, :-1])
         Test_dummie = pd.concat([Test_dummie, Test["FLT"]], axis = 1)

In [12]: Test_dummie.head()
```

Out[12]:

	SDT_DY_금	SDT_DY_목	SDT_DY_수	SDT_DY_월	SDT_DY_일	SDT_DY_토	SDT_DY_화	ARP_ARP1	ARP_ARP10	ARP_ARP11	...	FLO_F	FLO_G	FLO_H	FLO_I	FL
0	0	0	0	0	1	0	0	1	0	0	...	0	0	0	0	
1	0	0	0	0	1	0	0	1	0	0	...	0	0	0	0	
2	0	0	0	0	1	0	0	1	0	0	...	0	0	0	0	
3	0	0	0	0	1	0	0	1	0	0	...	1	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	

5 rows × 52 columns

머신러닝을 돌리기 위한 형식을 맞추는 과정

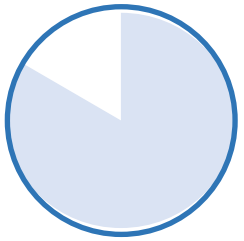
LabelEncoder , One-hot-encoding 등의 기술을 사용하여 전처리 진행

최적의 모델 선정

KneighborsClassifier

- 직관적인 이해 가능
- 학습시간 짧음
- 메모리 소모 큼
- 총 소요시간 김

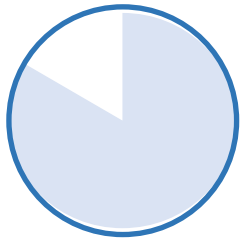
87.7%



DecisionTree

- 이해하기 쉬운 시각화
- 스케일에 구애 X
- 과대적합되는 경향
- 일반화 성능 안 좋음

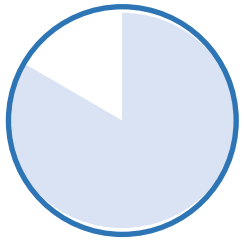
87.6%



RandomForestClassifier

- 매개변수 조정 X
- 코어 비례해서 속도 Up
- 고차원 데이터 X
- 과대적합되는 경향

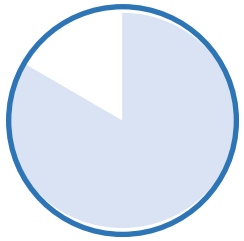
87.6%



GradientBoostingClassifier

- 스케일을 조정 X
- 연속적인 특성 가능
- 학습시간이 김
- 매개변수 조정 O

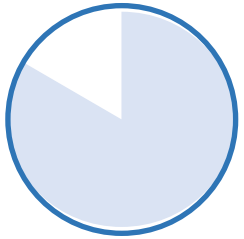
87.7%



MLPClassifier

- 데이터의 정보 추출
- 성능이 좋다
- 학습시간이 김
- 세밀한 전처리 과정

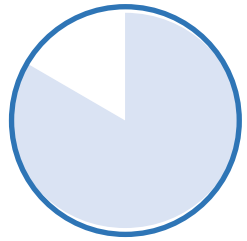
87.6%



Perceptron

- multi layer = MLP
- MLP의 기본 베이스
- MLP보다 성능 X
- 단계별 패턴 훈련 불가

87.5%



최적의 모델 선정

최적의 파라미터 찾기

```
knn = KNeighborsClassifier()
x = [ x for x in range(1, 100, 3)]
neighbors = {'n_neighbors' : x }
grid_searchKNN = GridSearchCV(knn, neighbors)
grid_searchKNN.fit(x_train, y_train)
```

```
print(grid_searchKNN.score(x_test, y_test))
print(grid_searchKNN.best_params_)
print(grid_searchKNN.best_score_)
```

```
0.8765470096546362
{'n_neighbors': 88}
0.8769001904746477
```

GridSearch 를 이용하여 해당 모델의 가장 적합한 파라미터를 찾음

Cross_val_score을 통한 검증

```
score_knn = cross_val_score(knn, Test_Concat.iloc[:987709, :-1], Test_Concat.iloc[:987709, -1])
score_tree = cross_val_score(tree, Test_Concat.iloc[:987709, :-1], Test_Concat.iloc[:987709, -1])
score_gbrt = cross_val_score(gbrt, Test_Concat.iloc[:987709, :-1], Test_Concat.iloc[:987709, -1])
score_rfP = cross_val_score(rfP, Test_Concat.iloc[:987709, :-1], Test_Concat.iloc[:987709, -1])
score_rf = cross_val_score(rf, Test_Concat.iloc[:987709, :-1], Test_Concat.iloc[:987709, -1])
```

```
print(score_knn.mean(), score_tree.mean(), score_gbrt.mean(), score_rfP.mean(), score_rf.mean())
```

```
0.8558664541379937 0.870546892149822 0.8742787601711505 0.7630852868724406 0.8703788267270198
```

Cross_val_score 을 통해 여러 가지 train 값에 대한 검증을 실시함.

최적의 모델 선정

Proba를 이용한 가치있는 데이터 찾기

```
Knn_probaR = knn.predict_proba(Test_Concat.iloc[987709:, :-1])
Rf_probaR = rf.predict_proba(Test_Concat.iloc[987709:, :-1])
Tree_probaR = tree.predict_proba(Test_Concat.iloc[987709:, :-1])
Gbrt_probaR = gbrt.predict_proba(Test_Concat.iloc[987709:, :-1])
```

```
Tree_probaRT = Tree_probaR[1].map(lambda x = 1 : x ==1 )

Tree_probaRT = Tree_probaRT.map({False: 0, True : 1})

Rf_probaRT = Rf_probaR[1].map(lambda x = 1 : x > 0.99)

Rf_probaRT = Rf_probaRT.map({False: 0, True : 1})

Knn_probaRT = Knn_probaR[1].map(lambda x = 1 : x == 1)

Knn_probaRT = Knn_probaRT.map({False: 0, True : 1})

Gbrt_probaRT = Gbrt_probaR[1].map(lambda x = 1 : x >0.46)

Gbrt_probaRT = Gbrt_probaRT.map({False: 0, True : 1})

Total_probaR = Tree_probaRT + Rf_probaRT + Knn_probaRT + Gbrt_probaRT
```

Proba 함수를 통해 해당 모델에서 가장 정확한 값을 산출함.

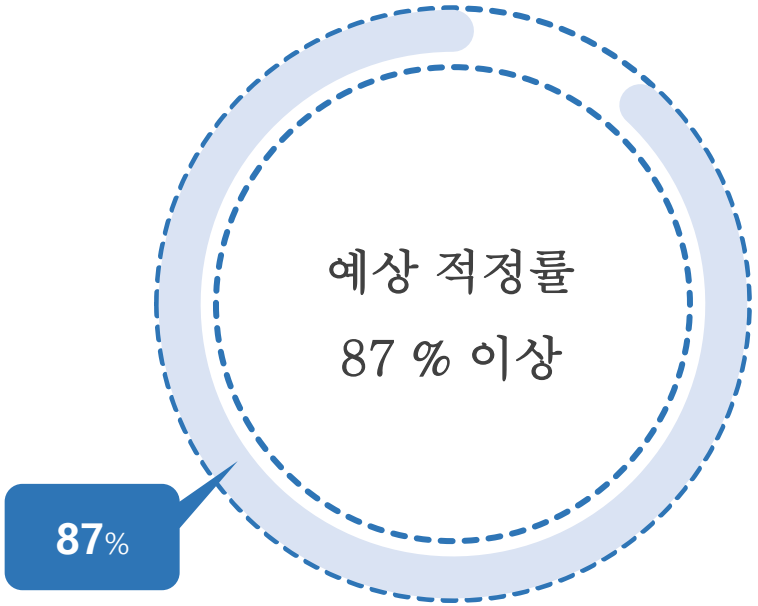
Predict를 통한 정답지 추출

```
Knn_predict = knn.predict(Test_Concat.iloc[987709:, :-1])
Tree_predict = tree.predict(Test_Concat.iloc[987709:, :-1])
Gbrt_predict = gbrt.predict(Test_Concat.iloc[987709:, :-1])
RfP_predict = rfP.predict(Test_Concat.iloc[987709:, :-1])
Rf_predict = rf.predict(Test_Concat.iloc[987709:, :-1])
```

```
Total_predict = Knn_predict + Tree_predict + Gbrt_predict + RfP_predict + Rf_predict
```

모델이 예측한 정답지를 모두 합쳐서 예측 정답지를 만듬

모델 적용



Predict Data 100%

Predict를 100% 기준

Proba Data 150%

Proba에 150%의 가산치를 줌

최종 예측 답안지

지연 여부는 최종 예측 답안지 X > 2 , 'Y'

지연 확률은 최종 예측 답안지 X * 12; X <100

Big Contest Futures League

Thank You