자연어 기반 인공지능 산업분류 자동화 경진대회

대회소개

통계데이터 인공지능 활용대회란?

- 통계 데이터의 새로운 활용 분야를 발굴하고 데이터의 잠재적 가치를 제고하고자,
  자연어 기반의 인공지능 분류 학습에 필요한 양질의 통계데이터를 개방하여,
  이를 통해 인공지능 학습 및 활용 저변을 확대하고자 개최하는 대회

지원 대상

- 통계데이터와 인공지능(AI) 및 기계학습에 관심 있는 대한민국 국적의 누구나 지원 가능

※개인 또는 팀(3명 이하)의 내국인, 외국인의 경우 개인 참여는 어려우며 한국인이 포함된 팀의 팀원으로 참가 가능

공모 주제

- **"자연어 기반 인공지능 산업분류 자동화"**
  - 자연어 기반의 통계데이터를 인공지능으로 자동 분류하는 기계학습 모델 발굴로 통계 데이터 활용 저변 확대

주관 및 주최 : 한국통계진흥원

Contents

01 경진대회 소개

대회소개

통계데이터 인공지능 활용대회란?

- 통계 데이터의 새로운 활용 분야를 발굴하고 데이터의 잠재적 가치를 제고하고자,
  자연어 기반의 인공지능 분류 학습에 필요한 양질의 통계데이터를 개방하여,
  이를 통해 인공지능 학습 및 활용 저변을 확대하고자 개최하는 대회

지원 대상

- 통계데이터와 인공지능(AI) 및 기계학습에 관심 있는 대한민국 국적의 누구나 지원 가능

※개인 또는 팀(3명 이하)의 내국인, 외국인의 경우 개인 참여는 어려우며 한국인이 포함된 팀의 팀원으로 참가 가능

공모 주제

- **"자연어 기반 인공지능 산업분류 자동화"**
  - 자연어 기반의 통계데이터를 인공지능으로 자동 분류하는 기계학습 모델 발굴로 통계 데이터 활용 저변 확대

01 경진대회 소개

진행일정

참가신청	자료 분석 기간	결과물 제출	심사 및 결과,시상
3.7.(월) ~ 3.21.(월)	3.14.(월) ~ 4.13.(수)		4.14.(목) ~ 5.4.(수)

심사방법

- 정확도 평가 및 전문가 평가
- (1차 평가) 예측값의 정확도 기준 시상 인원의 2~3배수 선출
- (2차 평가) 1차 심사 대상자의 코드설명자료로 코드작성의 적정성 평가

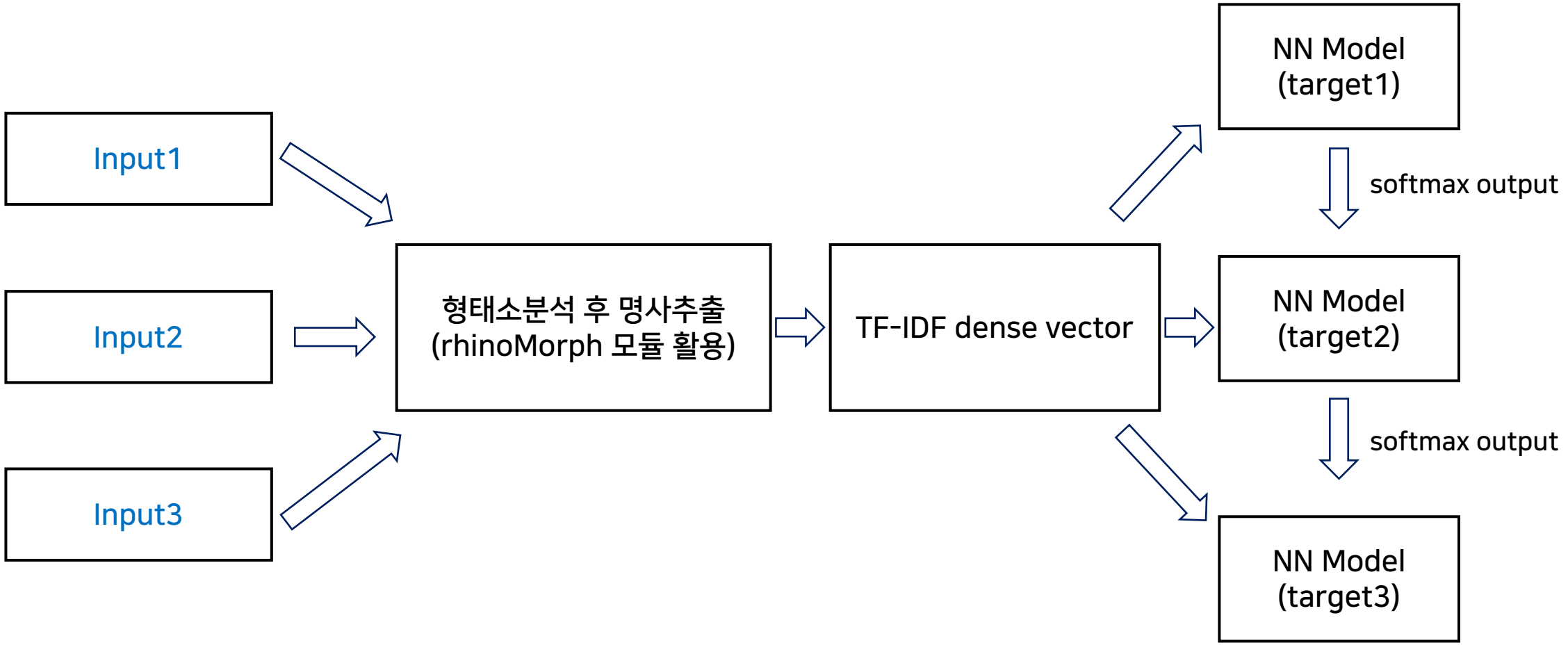
01 경진대회 소개 – 주요 특징

- ✓ 3개의 feature 이용해 3가지 target을 각각 분류하는 문제
- ✓ 각 targe은 의존성이 존재 (target1 ⊃ target2 ⊃ target3)

각 target 간 의존성을 고려한 모델설계가 필요

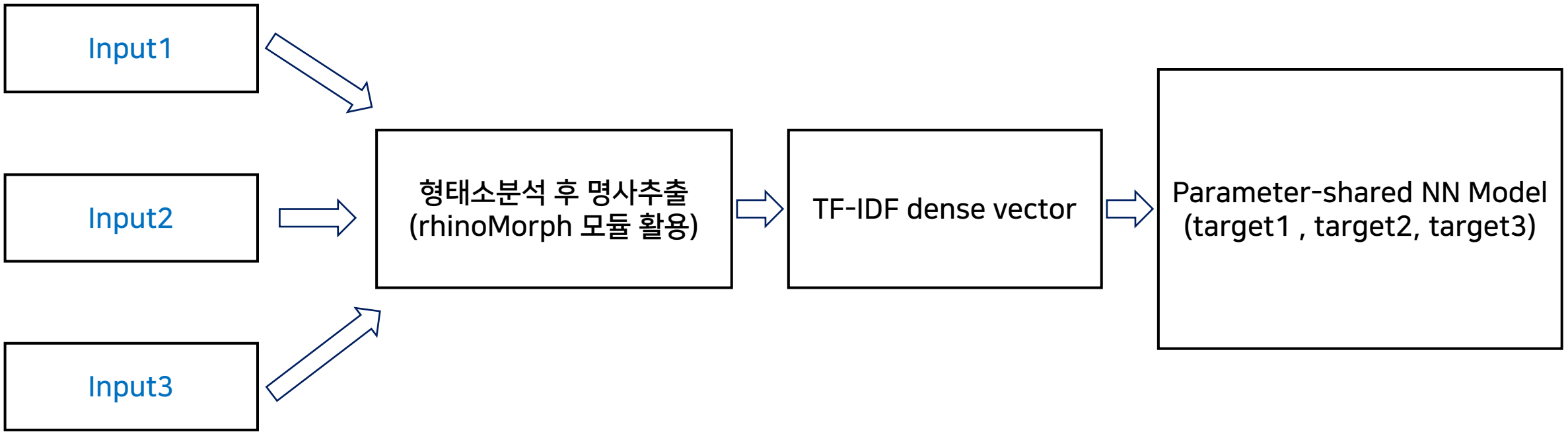
02 아키텍쳐 개요1 – Stage Learning

✓ **각 target에 대한 모델을 따로 학습시키고, 의존성이 있는 target의 모델은 의존된 target을 학습시킨 모델의 최종 output을 input에 결합하여 학습**



02 아키텍쳐 개요2 – Multi Task Learning

✓ **파라미터를 공유하는 모델을 생성하여 학습**
(Tensorflow Recommendation System 내 예시 코드를 수정하여 활용)



03 결과 정리

순위현황 *(22.06.08. 기준)

현재 순위	응시 번호	Accuracy	F1-score
1등	AI0149	91.33	81.57
2등	AI0135	91.26	82.31
3등	AI0288	91.25	81.31

* 최종 심사결과와 상이할 수 있음.

나의점수 *(22.06.08. 기준)

순위	응시 번호	Accuracy	F1-score
59등	AI0184	88.38	76.67

* 순위현황, 나의 점수: 휴일은 집계되지 않음.

- ✓ Multi Task Learning 방식으로 Accracy 88.38 달성
- ✓ 3가지 targe에 대해 하나의 모델로만 학습하여 상위권 score와 필적할 만한 성능을 보임
- ✓ 적은 파라미터로 높은 performance의 모델을 설계

감사합니다

04 Appendix - 구현코드

```python
class task_classifier(tf.keras.layers.Layer, base.Task):
    def __init__(
        self,
        loss,
        metrics=None,
        prediction_metrics=None,
        label_metrics=None,
        loss_metrics=None,
        name=None):
        super().__init__(name=name)

        self._loss = (loss if loss is not None else tf.keras.losses.BinaryCrossentropy())
        self._ranking_metrics = metrics or []
        self._prediction_metrics = prediction_metrics or []
        self._label_metrics = label_metrics or []
        self._loss_metrics = loss_metrics or []

    def call(
        self,
        labels,
        predictions,
        sample_weight=None,
        training=False,
        compute_metrics=True):
        loss = self._loss(y_true=labels, y_pred=predictions, sample_weight=sample_weight)
        if not compute_metrics:
            return loss
        update_ops = []

        for metric in self._ranking_metrics:
            update_ops.append(metric.update_state(
                y_true=labels, y_pred=predictions, sample_weight=sample_weight))

        for metric in self._prediction_metrics:
            update_ops.append(
                metric.update_state(predictions, sample_weight=sample_weight))

        for metric in self._label_metrics:
            update_ops.append(
                metric.update_state(labels, sample_weight=sample_weight))

        for metric in self._loss_metrics:
            update_ops.append(
                metric.update_state(loss, sample_weight=sample_weight))

        # Custom metrics may not return update ops, unlike built-in
```

```python
class Model_Mutlitask(tfrs.models.Model):
    def __init__(self, model_digit_base, loss_weight=(1.0, 1.0, 1.0)):
        # We take the loss weights in the constructor: this allows us to instantiate
        # several model objects with different loss weights.
        super().__init__()
        self.model_digit_base = model_digit_base
        # self.model_digit1 = create_classifier(len(target_encoder.dic_cat["digit_1"].keys()))
        # self.model_digit2 = create_classifier(len(target_encoder.dic_cat["digit_2"].keys()))
        # self.model_digit3 = create_classifier(len(target_encoder.dic_cat["digit_3"].keys()))
        self.task_digit1 = task_classifier(
            loss=tf.keras.losses.CategoricalCrossentropy(name="digit1_loss"),
            metrics=[tf.keras.metrics.CategoricalCrossentropy(name="digit1_logloss"),
                     tf.keras.metrics.CategoricalAccuracy(name="digit1_acc"),
                     tfa.metrics.F1Score(num_classes=len(target_encoder.dic_cat["digit_1"].keys()), average="
        )
        self.task_digit2 = task_classifier(
            loss=tf.keras.losses.CategoricalCrossentropy(name="digit2_loss"),
            metrics=[tf.keras.metrics.CategoricalCrossentropy(name="digit2_logloss"),
                     tf.keras.metrics.CategoricalAccuracy(name="digit2_acc"),
                     tfa.metrics.F1Score(num_classes=len(target_encoder.dic_cat["digit_2"].keys()), average="
        )
        self.task_digit3 = task_classifier(
            loss=tf.keras.losses.CategoricalCrossentropy(name="digit3_loss"),
            metrics=[tf.keras.metrics.CategoricalCrossentropy(name="digit3_logloss"),
                     tf.keras.metrics.CategoricalAccuracy(name="digit3_acc"),
                     tfa.metrics.F1Score(num_classes=len(target_encoder.dic_cat["digit_3"].keys()), average="
        )


        self.weighted_metrics = task_weighted_metric(
            metrics=[tf.keras.metrics.Mean(name="weighted_logloss"),
                     tf.keras.metrics.Mean(name="weighted_acc"),
                     tf.keras.metrics.Mean(name="weighted_f1score")]
        )
        self.loss_weight = loss_weight
    def compute_loss(self, features, training=False):
        softmax_digit1, softmax_digit2, softmax_digit3 = self.model_digit_base(features[0])
        loss_digit1 = self.task_digit1(predictions=softmax_digit1, labels=features[1][0], sample_weight=None)
        loss_digit2 = self.task_digit2(predictions=softmax_digit2, labels=features[1][1], sample_weight=None)
        loss_digit3 = self.task_digit3(predictions=softmax_digit3, labels=features[1][2], sample_weight=None)
        raw_scores = [
            (
                self.task_digit1._ranking_metrics[0].result(),
                self.task_digit2._ranking_metrics[0].result()
```