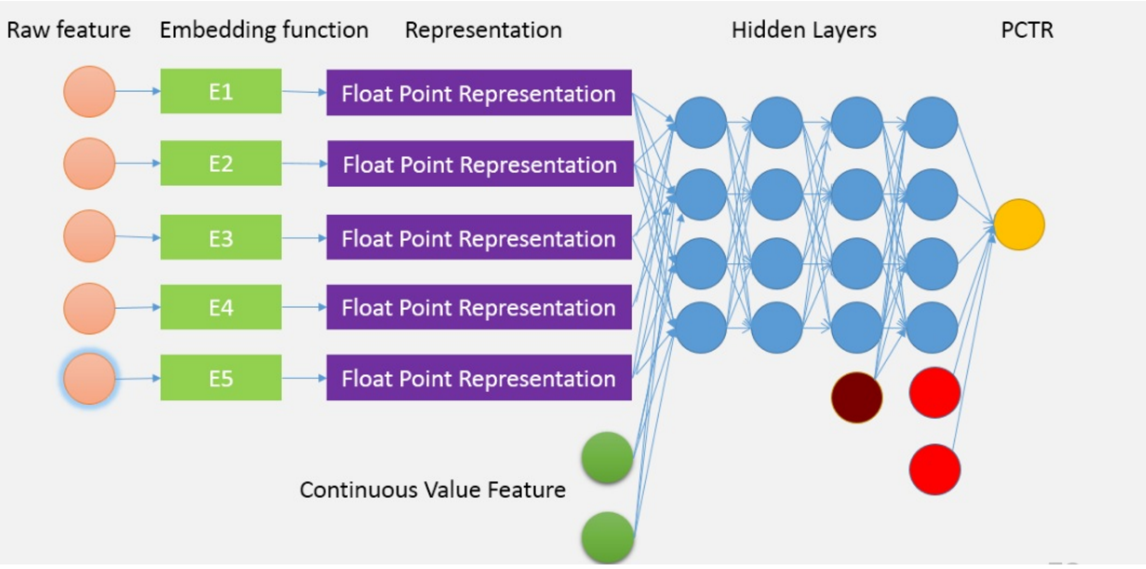
Wide & Deep

-
  - 0、贴在前言
  - 1、相关资料
  - 2、数学原理
    - 2.1 一句话总结
    - 2.2 What is Wide & Deep
    - 2.3 如何用Wide & Deep 玩转Recommender system
    - 2.4 Recommender system pipeline overview
    - 2.5 How Does Google Play Wide & Deep in Recommender system?
    - 2.6 模型详解
  - 3、 TensorFlow实现

0、贴在前言

回顾DNN CTR Prediction模型的设计， 综合学术界以及工业界的经验， 整个模型的设计思路大致如下：



1、相关资料

- 英文paper：https://arxiv.org/pdf/1606.07792.pdf
- 中文资料：
  - 理论：https://zhuanlan.zhihu.com/p/34676942
  - 实现：https://blog.csdn.net/m0_37744293/article/details/69950262

2、数学原理

2.1 一句话总结

> - **从数学分析 Wide & Deep**，
>   - Wide 组件，是基于带交互的LR；
>   - Deep 组件，是DNN 模型；

- **从业务分析 Wide & Deep**，Wide & Deep Model：
  - Wide部分（LR）通过Feature Cross 精细刻画场景；
  - Deep部分（DNN）则强调Generalization；

2.2 What is Wide & Deep

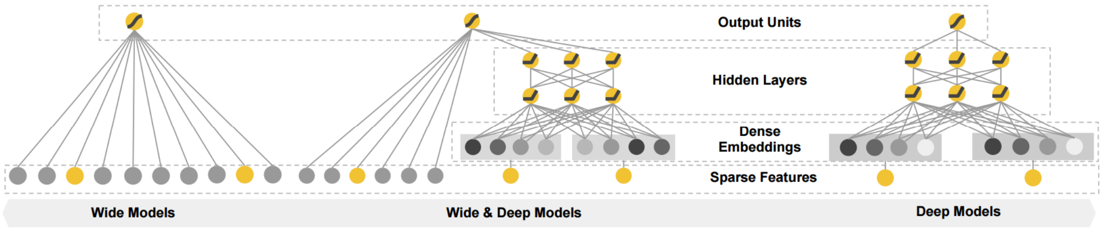


Figure 1: The spectrum of Wide & Deep models.

2.3 如何用Wide & Deep 玩转Recommender system

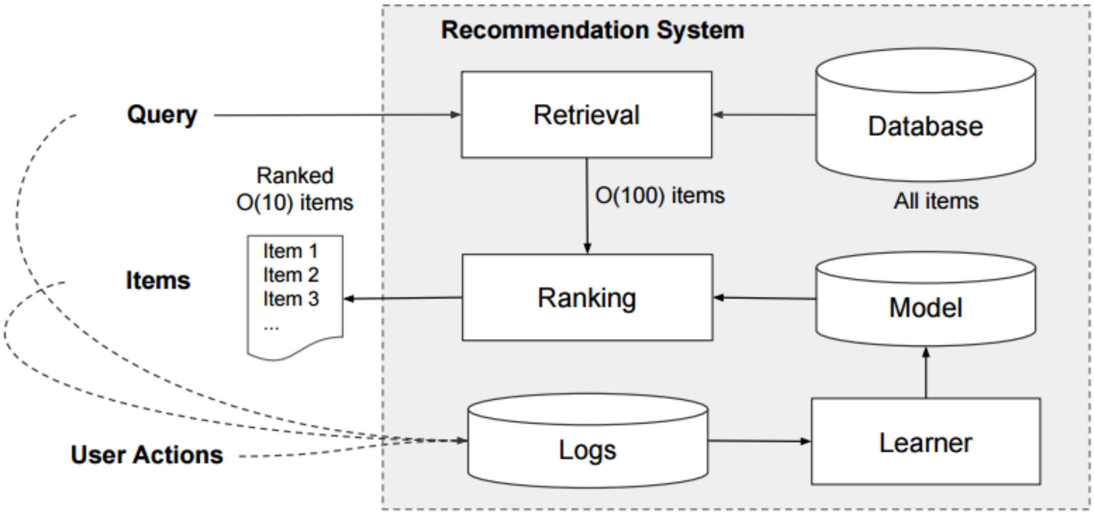


Figure 2: Overview of the recommender system.

2.4 Recommender system pipeline overview

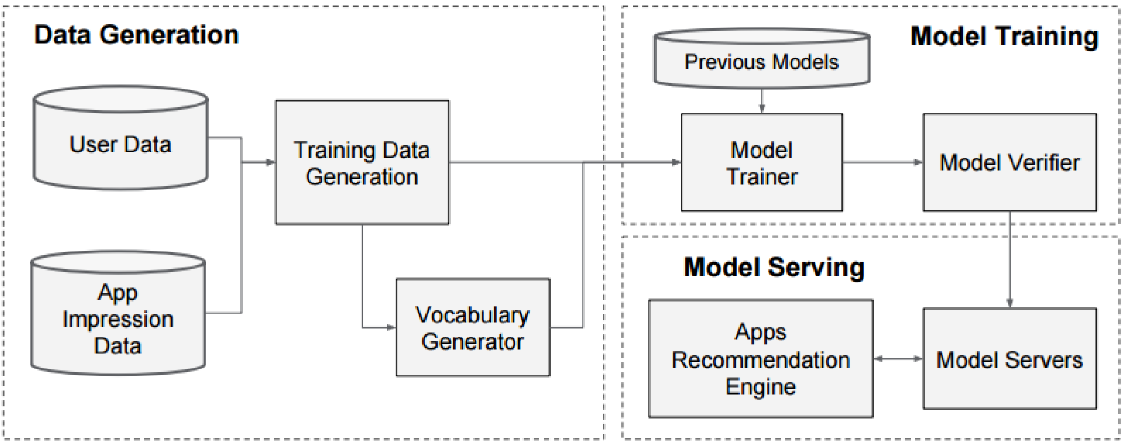


Figure 3: Apps recommendation pipeline overview.

2.5 How Does Google Play Wide & Deep in Recommender system?

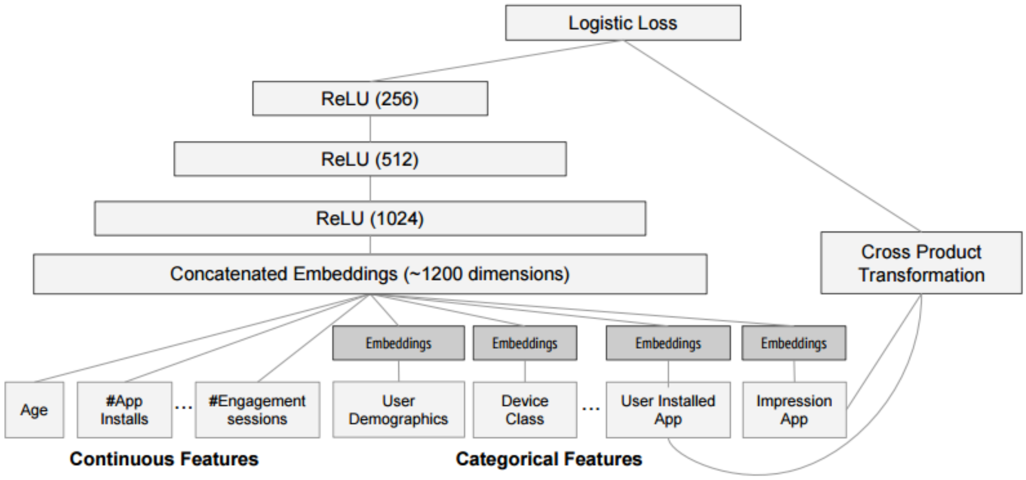
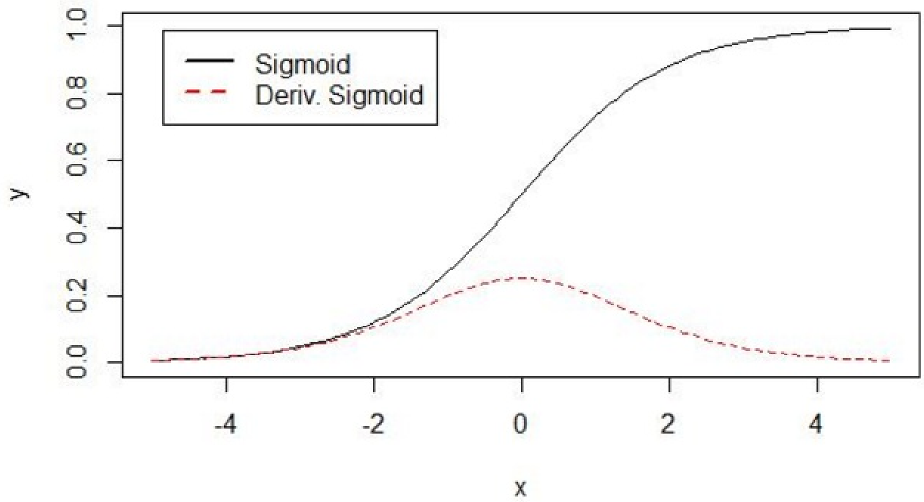
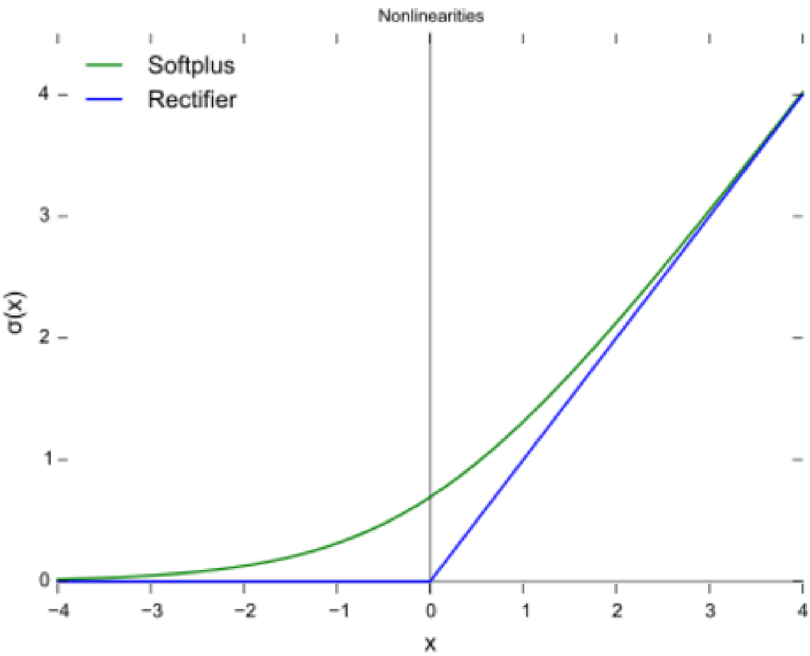


Figure 4: Wide & Deep model structure for apps recommendation.

2.6 模型详解

- **Wide**
  - Wide组件解决“**记忆性**”的问题：比如通过历史数据知道”麻雀会飞” ，” 鸽子会飞”。
  - 由一个LR模型构成，特征集包括原始特征和交互特征。这里有一个坑：看paper会误以为模型默认是对所有one-hot后的特征进行交互。**但看TensorFlow的实现代码会发现：哪些one-hot后的特征需要进行交互，和谁进行交互，做几次幂的交互，均由algo engineer指定。** ，具体公式如下：
    $\phi_k(x) = \prod_{i=1}^{d} x_i^{c_{ki}}, c_{ki} \in \{0, 1\}$
  - 如果历史数据里没有“钢铁侠会飞” ，那么单靠Wide组件就无法学到这部分知识。这是LR本身的缺陷，即**如果没有将某个交叉特征输入到模型，模型就学不到。**
  - 具体做法：
    - 将分类特征做完one-hot后再进行cross product。
    - 个人理解：
      - 这里一方面是利用类似于FM模型原理来增强分类特征的特征交互（co-occurrence）；**【这里只是一个局部FM，因为哪些feature进行交互，是由algo engineer指定。而FM是所有的feature进行交互，交互学习的方式是基于latent factor。】**
      - 另一方面是利用LR对高维稀疏特征的学习能力。
    - 作者把Wide Model所具备的能力称为“memorization”
- **Deep**
  - Deep组件解决“**泛化性**”的问题：从历史数据中从未见过的情形，比如”带翅膀的动物会飞”。
  - Deep 端对应的是 DNN 模型，每个特征对应一个低维的实数向量，我们称之为特征的**embedding**。DNN 模型通过反向传播（BP算法）调整隐藏层的权重，并且更新特征的 embedding。【其实整个 Wide & Deep Model 都是根据BP来更新参数的。】
  - Deep Model则是一个DNN，特征上除了原始特征还增加了分类特征的embedding，这个embedding在模型中属于独立的一层，embedding后的向量也是通过不断迭代学习出来的。

- 将高维稀疏分类特征映射到低维embedding特征这种方式有助于模型进行“generalization”。
- 每个 hideen layer都会做以下运算：
  $a^{l+1} = f(W^{(l)}a^{(l)} + b^{(l)})$
  其中l表示层数，f表示激活函数（通常为$ReLUs$），$a^{(l)}$表示第l层的$activations$，$b^{(l)}$表的$bias$，$W^{(l)}$表示第l层的$weights$
- **Why does DNN always use ReLUs?**
  - 一般我们优化参数时会用到误差反向传播算法，即要对激活函数求导，得到sigmoid函数的瞬时变化率，其导数表达式为：
    $\phi'(x) = \phi(x) \cdot (1 - \phi'(x))$
    对应的图形如下:
  - 

  - 由图可知，导数从0开始很快就又趋近于0了，易造成“梯度消失”现象，而ReLU的导数就不存在这样的问题，它的导数表达式如下：
    Relu函数的形状如下（蓝色）：
  - 

  - 对比sigmoid类函数，ReLUs的优势：
    1）单侧抑制；
    2）相对宽阔的兴奋边界；
    3）稀疏激活性；
- **为什么需要偏置量**

        - 需要在x接近0时，函数结果为其他值。
- **Wide & Deep**
    - 一个是通过特征交互关系来训练浅层模型，另一个则是通过特征在映射空间中的信息训练深层模型。
    - 模型预测正例的公式为：
      $P(Y=1|x) = \sigma(w_{wide}^{T}[x, \phi(x)] + w_{deep}^{T} a^{(l_f)} + b)$
      其中Y是二分类的label，$\sigma(\cdot)$是sigmoid function，$\phi(x)$是对原始特征x做cross product transformations，b是bias项。
      w_{wide}是所有$wide$模型权重向量，w_{deep}是应用在最终激活函数$a^{(l_f)}$上的权重。
    - **不需要pre-training，可以直接将sparse data 输入给模型。**
- **宏观感知**
    - 对于连续值的特征，是这么做归一化的，先将原始值通过累计分布函数进行变换P(X <=x),然后用分位数表示，最后用分位数进行归一。
    - 训练阶段，输入层将训练数据和词表产生稀疏和稠密的特征以及label。
    - Wide组件由用户安装app和展示app的交叉特征构成。
    - 对于deep组件，枚举类的稀疏特征embedding成为一个32维的向量，然后跟连续值特征拼接在一起成为一个1200维的向量。串联后的向量会灌进三层ReLU layer。
    - 最后deep组件和wide组件一起输入到logistic。
- **Train Model**
    - 这里有一个坑是：正确的最后一步是，wide组件和deep组件的输出的对数几率进行加权求和后，输入到共同的一个logistic loss function。paper原文内容：

      > **The wide component and deep component are combined using a weighted sum of their output log odds as the prediction, which is then fed to one common logistic loss function for joint training.**

    - 复习一下 logistic loss function:
      $cost(h_\theta(x), y) = -y_i log(h_\theta(x)) - (1-y_i) log(1-h_\theta(x))$
    - 模型训练的方式是“**joint training**”：
        - 同时优化所有参数，在训练阶段就考虑了Wide组件和Deep组件以及他们之间的加权求和的权重参数。
        - Wide组件仅需要补充Deep组件的弱点，通常是一小部分交叉特征的变换，而不是一个整个full-size的Wide模型。
        - paper原文内容：
          In comparison, for joint training the wide part only needs to complement the weaknesses of the deep part with a small number of cross-product feature transformations, rather than a full-size wide model.
    - Wide & Deep Model，整个joint training过程为：
        - 宏观过程是基于BP算法来实现优化过程；
        - Wide部分，默认是使用带L1正则项的ftrl进行求解；
        - Deep部分，默认是使用AdaGrad Optimizer进行求解；
        - *PS：这里的optimizer最好自己指定。*
- 数据预处理

- Wide部分的输入特征：
  raw input features and transformed features [手挑的交叉特征].
  notice: W&D这里的cross-product transformation：
  只在离散特征之间做组合，不管是文本策略型的，还是离散值的；没有连续值特征的啥事，至少在W&D的paper里面是这样使用的。
  - Deep部分的输入特征： raw input+embedding处理
  对非连续值之外的特征做embedding处理，这里都是策略特征，就是乘以个embedding-matrix。在TensorFlow里面的接口是：tf.feature_column.embedding_column，默认trainable=True.
  - 对连续值特征的处理是：将其按照累积分布函数P(X≤x)，压缩至[0,1]内。
  - 累积分布的压缩很简单：数学意义是表示随机变量小于或等于其某一个取值x的概率。
  - 举个例子：
  - feature的范围为(0, 50),划分5个桶，每个桶内的落入的样本为[10, 4, 2, 5, 4],那么占比为[0.4,0.16,0.08,0.2,0.16]。
  - 如果一个feature落入第二个桶，那么归一化结果为(10+4)/25 = 0.56
  - 使用bucketized_column() 实现。
- notice: Wide部分用FTRL+L1来训练；Deep部分用AdaGrad来训练。

3、 **TensorFlow实现**

- 官方教程
- 比较好的总结，还包括了DCN的实现：https://blog.csdn.net/yujianmin1990/article/details/78989099

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import shutil
import sys

import tensorflow as tf

_CSV_COLUMNS = [
    'age', 'workclass', 'fnlwgt', 'education', 'education_num',
    'marital_status', 'occupation', 'relationship', 'race', 'gender',
    'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
    'income_bracket'
]

_CSV_COLUMN_DEFAULTS = [[0], [''], [0], [''], [0], [''], [''], [''], [''], ['']
,
                        [0], [0], [0], [''], ['']]

parser = argparse.ArgumentParser()

parser.add_argument(
    '--model_dir', type=str, default='./tmp/census_model',
    help='Base directory for the model.')
```

```
parser.add_argument(
    '--model_type', type=str, default='wide_deep',
    help="Valid model types: {'wide', 'deep', 'wide_deep'}.")

parser.add_argument(
    '--train_epochs', type=int, default=40, help='Number of training epochs.')

parser.add_argument(
    '--epochs_per_eval', type=int, default=2,
    help='The number of training epochs to run between evaluations.')

parser.add_argument(
    '--batch_size', type=int, default=40, help='Number of examples per batch.')

parser.add_argument(
    '--train_data', type=str, default='./tmp/census_data/adult.data',
    help='Path to the training data.')

parser.add_argument(
    '--test_data', type=str, default='./tmp/census_data/adult.test',
    help='Path to the test data.')

_NUM_EXAMPLES = {
    'train': 32561,
    'validation': 16281,
}


def build_model_columns():
  """Builds a set of wide and deep feature columns."""
  # Continuous columns
  age = tf.feature_column.numeric_column('age')
  education_num = tf.feature_column.numeric_column('education_num')
  capital_gain = tf.feature_column.numeric_column('capital_gain')
  capital_loss = tf.feature_column.numeric_column('capital_loss')
  hours_per_week = tf.feature_column.numeric_column('hours_per_week')

  education = tf.feature_column.categorical_column_with_vocabulary_list(
      'education', [
          'Bachelors', 'HS-grad', '11th', 'Masters', '9th', 'Some-college',
          'Assoc-acdm', 'Assoc-voc', '7th-8th', 'Doctorate', 'Prof-school',
          '5th-6th', '10th', '1st-4th', 'Preschool', '12th'])

  marital_status = tf.feature_column.categorical_column_with_vocabulary_list(
      'marital_status', [
          'Married-civ-spouse', 'Divorced', 'Married-spouse-absent',
          'Never-married', 'Separated', 'Married-AF-spouse', 'Widowed'])

  relationship = tf.feature_column.categorical_column_with_vocabulary_list(
      'relationship', [
          'Husband', 'Not-in-family', 'Wife', 'Own-child', 'Unmarried',
          'Other-relative'])

  workclass = tf.feature_column.categorical_column_with_vocabulary_list(
      'workclass', [
          'Self-emp-not-inc', 'Private', 'State-gov', 'Federal-gov',
```

```
        'Local-gov', '?', 'Self-emp-inc', 'Without-pay', 'Never-worked'])

  # To show an example of hashing:
  occupation = tf.feature_column.categorical_column_with_hash_bucket(
      'occupation', hash_bucket_size=1000)

  # Transformations.
  age_buckets = tf.feature_column.bucketized_column(
      age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])

  # Wide columns and deep columns.
  base_columns = [
      education, marital_status, relationship, workclass, occupation,
      age_buckets,
  ]

  crossed_columns = [
      tf.feature_column.crossed_column(
          ['education', 'occupation'], hash_bucket_size=1000),
      tf.feature_column.crossed_column(
          [age_buckets, 'education', 'occupation'], hash_bucket_size=1000),
  ]

  wide_columns = base_columns + crossed_columns

  deep_columns = [
      age,
      education_num,
      capital_gain,
      capital_loss,
      hours_per_week,
      tf.feature_column.indicator_column(workclass),
      tf.feature_column.indicator_column(education),
      tf.feature_column.indicator_column(marital_status),
      tf.feature_column.indicator_column(relationship),
      # To show an example of embedding
      tf.feature_column.embedding_column(occupation, dimension=8),
  ]

  return wide_columns, deep_columns


def build_estimator(model_dir, model_type):
  """Build an estimator appropriate for the given model type."""
  wide_columns, deep_columns = build_model_columns()
  hidden_units = [100, 75, 50, 25]

  # Create a tf.estimator.RunConfig to ensure the model is run on CPU, which
  # trains faster than GPU for this model.
  run_config = tf.estimator.RunConfig().replace(
      session_config=tf.ConfigProto(device_count={'GPU': 0}))

  if model_type == 'wide':
    return tf.estimator.LinearClassifier(
        model_dir=model_dir,
        feature_columns=wide_columns,
```

```
        config=run_config)
  elif model_type == 'deep':
    return tf.estimator.DNNClassifier(
        model_dir=model_dir,
        feature_columns=deep_columns,
        hidden_units=hidden_units,
        config=run_config)
  else:
    return tf.estimator.DNNLinearCombinedClassifier(
        model_dir=model_dir,
        linear_feature_columns=wide_columns,
        dnn_feature_columns=deep_columns,
        dnn_hidden_units=hidden_units,
        config=run_config)


def input_fn(data_file, num_epochs, shuffle, batch_size):
  """Generate an input function for the Estimator."""
  assert tf.gfile.Exists(data_file), (
      '%s not found. Please make sure you have either run data_download.py or '
      'set both arguments --train_data and --test_data.' % data_file)

  def parse_csv(value):
    print('Parsing', data_file)
    columns = tf.decode_csv(value, record_defaults=_CSV_COLUMN_DEFAULTS)
    features = dict(zip(_CSV_COLUMNS, columns))
    labels = features.pop('income_bracket')
    return features, tf.equal(labels, '>50K')

  # Extract lines from input files using the Dataset API.
  dataset = tf.data.TextLineDataset(data_file)

  if shuffle:
    dataset = dataset.shuffle(buffer_size=_NUM_EXAMPLES['train'])

  dataset = dataset.map(parse_csv, num_parallel_calls=5)

  # We call repeat after shuffling, rather than before, to prevent separate
  # epochs from blending together.
  dataset = dataset.repeat(num_epochs)
  dataset = dataset.batch(batch_size)

  iterator = dataset.make_one_shot_iterator()
  features, labels = iterator.get_next()
  return features, labels


def main(unused_argv):
  # Clean up the model directory if present
  shutil.rmtree(FLAGS.model_dir, ignore_errors=True)
  model = build_estimator(FLAGS.model_dir, FLAGS.model_type)

  # Train and evaluate the model every `FLAGS.epochs_per_eval` epochs.
  for n in range(FLAGS.train_epochs // FLAGS.epochs_per_eval):
    model.train(input_fn=lambda: input_fn(
        FLAGS.train_data, FLAGS.epochs_per_eval, True, FLAGS.batch_size))
```

```
    results = model.evaluate(input_fn=lambda: input_fn(
        FLAGS.test_data, 1, False, FLAGS.batch_size))

    # Display evaluation metrics
    print('Results at epoch', (n + 1) * FLAGS.epochs_per_eval)
    print('-' * 60)

    for key in sorted(results):
      print('%s: %s' % (key, results[key]))

  '''Export Trained Model for Serving'''
  wideColumns, DeepColumns = build_model_columns()
  feature_columns = DeepColumns
  feature_spec = tf.feature_column.make_parse_example_spec(feature_columns)
  export_input_fn = tf.estimator.export.build_parsing_serving_input_receiver_fn
(feature_spec)
  servable_model_dir = "./tmp/census_exported"
  servable_model_path = model.export_savedmodel(servable_model_dir, export_inpu
t_fn)
  print("*********** Done Exporting at PAth - %s", servable_model_path )



if __name__ == '__main__':
  tf.logging.set_verbosity(tf.logging.INFO)
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```