JaneStreet Market Prediction (Kaggle)

1. 赛题描述

根据股票历史特征和对应的收益状况，预测股票未来收益 (二分类)。

2. 数据探索

本次比赛是由 Kaggle 举办的 Kernel赛，我们可以看到训练集，但无法看到榜单的测试集。

```
data.shape # output: (2390491, 137)
data.head()
```

ts_id	date	weight	resp_1	resp_2	resp_3	resp_4	resp	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6
0	0	0.000000	0.009916	0.014079	0.008773	0.001390	0.006270	1	-1.872746	-2.191242	-0.474163	-0.323046	0.014688	-0.002484
1	0	16.673515	-0.002828	-0.003226	-0.007319	-0.011114	-0.009792	-1	-1.349537	-1.704709	0.068058	0.028432	0.193794	0.138212
2	0	0.000000	0.025134	0.027607	0.033406	0.034380	0.023970	-1	0.812780	-0.256156	0.806463	0.400221	-0.614188	-0.354800
3	0	0.000000	-0.004730	-0.003273	-0.000461	-0.000476	-0.003200	-1	1.174378	0.344640	0.066872	0.009357	-1.006373	-0.676458
4	0	0.138531	0.001252	0.002165	-0.001215	-0.006219	-0.002604	1	-3.172026	-3.093182	-0.161518	-0.128149	-0.195006	-0.143780

feature_118	feature_119	feature_120	feature_121	feature_122	feature_123	feature_124	feature_125	feature_126	feature_127	feature_128	feature_129
6.649580	-1.472686	NaN	NaN	1.168391	8.313583	1.782433	14.018213	2.653056	12.600292	2.301488	11.445807
1.612348	-1.664544	NaN	NaN	-1.178850	1.777472	-0.915458	2.831612	-1.417010	2.297459	-1.304614	1.898684
1.817895	-2.432424	NaN	NaN	6.115747	9.667908	5.542871	11.671595	7.281757	10.060014	6.638248	9.427299
0.099560	-2.485993	NaN	NaN	2.838853	0.499251	3.033732	1.513488	4.397532	1.266037	3.856384	1.013469
2.134804	-1.279284	NaN	NaN	0.344850	4.101145	0.614252	6.623456	0.800129	5.233243	0.362636	3.926633

图 1

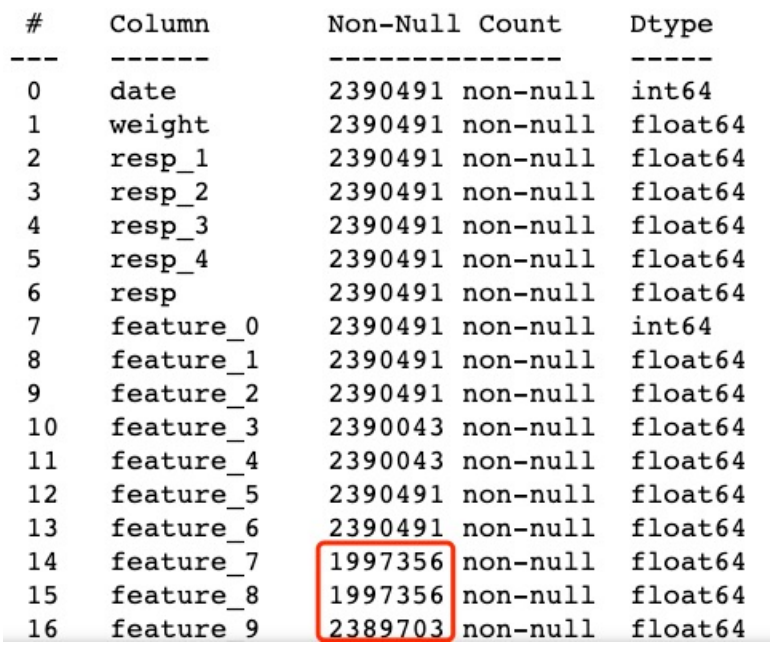
- date 交易日期
- weight 交易权重
- resp 交易回报 (用于计算比赛结果的评价指标)

  resp 1-4 交易短期、中期、长期、超长期回报，与resp密切相关
- feature 0-129 130个的匿名特征。

注：测试集中没有与交易回报相关的5个属性 (resp、resp 1-4)，其他属性与训练集中一致。测试集中每一条记录都可以看作一个交易机会，需要根据属性数据预测交易行为 (action) 是否应该发生 (action=1表示发生，0表示不发生)。最终的评价函数为：

$$p = \sum weight_i \cdot resp_i \cdot action_i \tag{1}$$

```
data.info(show_counts=True)
```



#	Column	Non-Null Count	Dtype
0	date	2390491 non-null	int64
1	weight	2390491 non-null	float64
2	resp_1	2390491 non-null	float64
3	resp_2	2390491 non-null	float64
4	resp_3	2390491 non-null	float64
5	resp_4	2390491 non-null	float64
6	resp	2390491 non-null	float64
7	feature_0	2390491 non-null	int64
8	feature_1	2390491 non-null	float64
9	feature_2	2390491 non-null	float64
10	feature_3	2390043 non-null	float64
11	feature_4	2390043 non-null	float64
12	feature_5	2390491 non-null	float64
13	feature_6	2390491 non-null	float64
14	feature_7	1997356 non-null	float64
15	feature_8	1997356 non-null	float64
16	feature_9	2389703 non-null	float64

图 2

所有属性均为数值类型，某些属性存在较多缺失值 (红框)。

```
1 data.describe()
```

	date	weight	resp_1	resp_2	resp_3	resp_4	resp	feature_0	feature_1	feature_2	feature_3	feature_4
count	2390491.00	2390491.00	2390491.00	2390491.00	2390491.00	2390491.00	2390491.00	2390491.00	2390491.00	2390491.00	2390043.00	2390043.00
mean	247.87	3.03	0.00	0.00	0.00	0.00	0.00	0.01	0.39	0.36	0.01	0.00
std	152.27	7.67	0.01	0.01	0.02	0.03	0.03	1.00	2.56	2.48	1.94	1.75
min	0.00	0.00	-0.37	-0.53	-0.57	-0.60	-0.55	-1.00	-3.17	-3.09	-25.42	-19.12
25%	104.00	0.16	-0.00	-0.00	-0.01	-0.01	-0.01	-1.00	-1.30	-1.26	-1.02	-0.98
50%	254.00	0.71	0.00	0.00	0.00	0.00	0.00	1.00	-0.00	-0.00	0.00	0.00
75%	382.00	2.47	0.00	0.00	0.01	0.01	0.01	1.00	1.58	1.53	1.05	0.99
max	499.00	167.29	0.25	0.29	0.33	0.51	0.45	1.00	74.43	148.08	25.87	19.42

图 3

匿名属性的75%分位数和最大值有较大差距，这可能是异常值；也可能是正常值，如更大的匿名属性值对应更大的resp。

```
1 data[data.feature_1>60]
```

ts_id	date	weight	resp_1	resp_2	resp_3	resp_4	resp	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5
41128	5	0.09	0.03	0.04	0.04	0.07	0.06	-1	73.16	42.43	-2.35	-1.25	-2.48
257064	40	0.08	0.04	0.05	0.04	0.06	0.06	1	74.43	67.12	1.29	0.92	1.56
299302	44	0.00	0.01	0.01	0.01	0.00	0.01	-1	64.17	86.92	-2.10	-2.35	-3.03
523104	85	14.19	0.02	0.02	0.01	-0.04	-0.03	-1	72.16	54.37	1.11	0.62	1.86
1328832	283	0.12	0.01	-0.01	-0.01	0.02	-0.00	1	66.85	66.28	-2.62	-2.29	-2.51
1847976	393	13.56	-0.01	-0.01	-0.02	-0.03	-0.02	-1	68.59	148.08	1.02	1.72	0.28
2172955	459	2.44	0.04	0.04	0.05	0.06	0.05	1	68.78	83.19	6.05	6.67	5.87
2231449	469	13.24	0.04	0.03	0.01	-0.00	0.00	1	72.62	135.34	-3.80	-5.85	-3.04

图 4

我们筛选出 feature_1 取值较大的记录，仔细观察它们的情况，如图4所示。这些 feature_1 取值较大的记录，其对应的交易收益 (resp) 并没有表现反常，因此考虑将这些记录视作异常值。

此外，可以发现当 feature_1 取值较大时，feature_2 取值也较大，说明**这两个属性存在很强的相关性**。

3. 数据预处理

3.1 缺失值

```
# 查看各个属性缺失值比例
missing_rate = data.isnull().sum()/data.shape[0]*100
pd.DataFrame(missing_rate).T
```

feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	feature_11
0.00	0.00	0.02	0.02	0.00	0.00	16.45	16.45	0.03	0.03	3.35

图 5

缺失值只存在于匿名属性，使用**均值填充** (经测试，使用均值填充比中位数效果更好一些)。

```
data.fillna(data.mean(),inplace=True)
```

3.2 异常值

为了避免数据分布被破坏，只去除了每个属性取值非常特殊 (99.99%分位数) 的样本。

```
quantile_num = data.quantile(q=0.9999)
pd.DataFrame(quantile_num).T
```

feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8
1.0	29.819824	33.153558	13.756857	12.010048	9.160501	10.609517	17.366574	23.821238

图 6

```
for i in range(1,130):
    col = 'feature_' + str(i)
    condition = quantile_num.loc[:,col].values[0]
    data = data[data[col]<condition]
```

3.3 date 属性

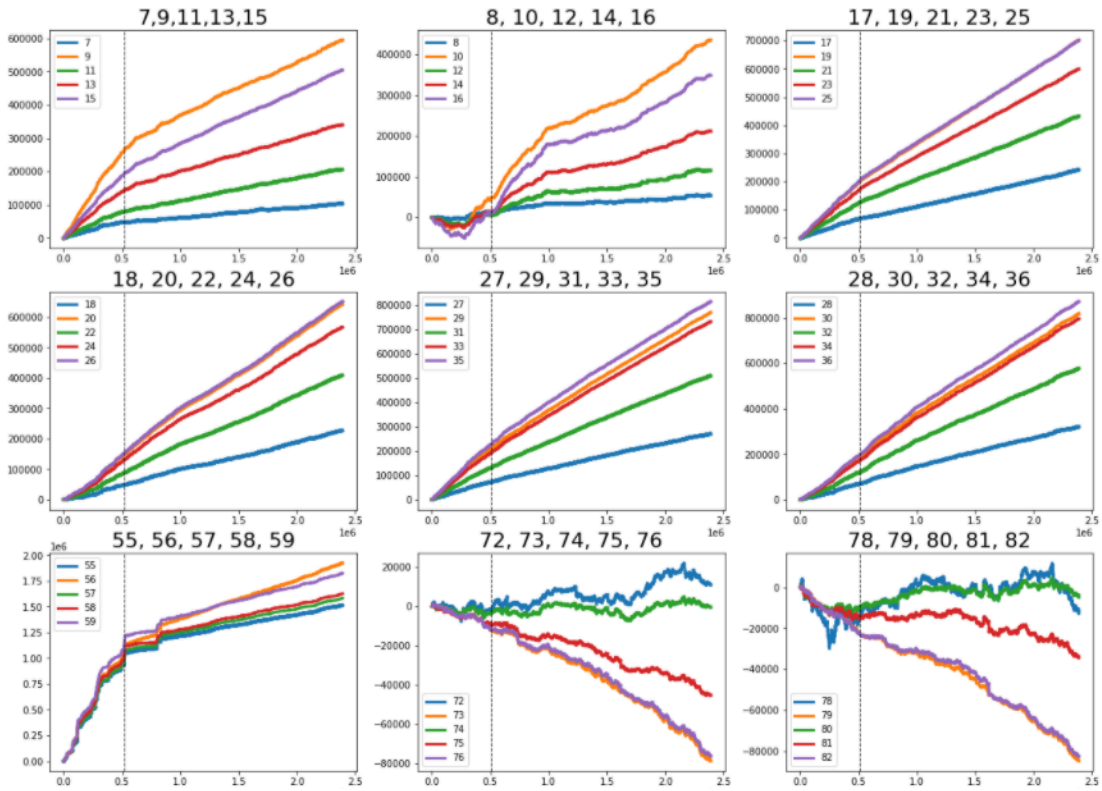


图 7

很多匿名属性在第85天之前和第85天之后 (上图中黑色竖线即为第85天) 趋势相差很大，我们仅保留了第85天之后的数据作为训练集，以让模型学习最近的数据趋势。(上图来自于Kaggle 论坛中的开源文章)

```
data = data.query('date > 85').reset_index(drop = True)
```

4. Baseline

虽然赛题的评价指标只使用 resp 计算，但是赛题官方提示了 resp1-4 与 resp有关。因此我们制定了一条 **策略**：使用130个匿名特征，对五个标签进行二分类预测问题。得到5个标签的预测值后，再对这五个标签应用投票法，确定最终action的预测结果。

我们选择全连接神经网络作为本题的Baseline方法，使用Keras实现。

首先使用Keras-Tuner库的贝叶斯调参方法搜索最佳的网络结构 (**隐含层数量** 和 **神经元个数**)。

```
# 选择前4/5数据作为训练集，后1/5数据作为验证集
train_data = data[data.date <= 420]
valid_data = data[data.date > 420]

X_train = train_data.loc[:, train_data.columns.str.contains('feature')]
X_valid = valid_data.loc[:, valid_data.columns.str.contains('feature')]
resp_cols = ['resp_1', 'resp_2', 'resp_3', 'resp_4', 'resp']
y_train = np.stack([(train_data[c] > 0).astype('int') for c in resp_cols]).T
y_valid = np.stack([(valid_data[c] > 0).astype('int') for c in resp_cols]).T
```

由于各个属性量纲基本一致，因此无需进行归一化处理。

```
def build_model(hp):
    inp = tf.keras.layers.Input(shape=(130,))
    x = tf.keras.layers.Dropout(0.2)(inp)
    for i in range(hp.Int('num_layers', 2, 5)):
        x = tf.keras.layers.Dense(units=hp.Int('units_' + str(i),min_value=50,max_value=400,step=50),activation='relu')(x)
        x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(5)(x)
    out = tf.keras.layers.Activation("sigmoid")(x)
    model = tf.keras.models.Model(inputs=inp, outputs=out)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(0.001),
        loss=['binary_crossentropy'],
        metrics=tf.keras.metrics.AUC(name="auc")
    )

    return model

tuner = BayesianOptimization(
    build_model,
    objective=Objective("val_auc", direction="max"),
    max_trials=10,   #总共试验10次，选十个参数配置
    executions_per_trial=1,
    directory='keras_tune_dir',
```

```
    project_name='tune_project1')

tuner.search(X_train, y_train,
             epochs=100,
             validation_data=(X_valid, y_valid),
             batch_size=5000)

tuner.results_summary()
```

最佳参数为 3个隐含层，神经元个数分别为 [300, 200, 50]。

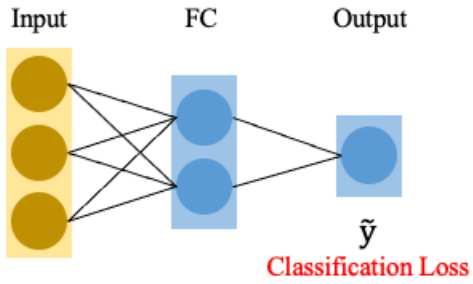


图 8

按照搜索出的网络结构训练模型，使用SKLearn库的TimeSeriesSplit方法对数据集进行三折划分，避免时间穿越：

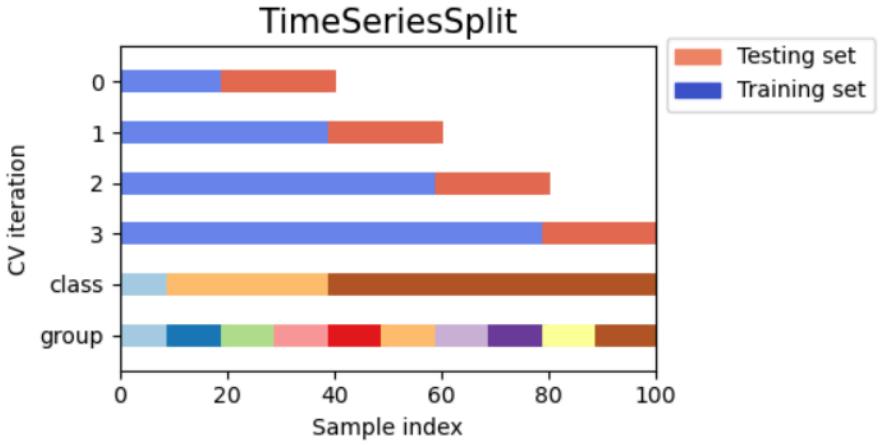


图 9

```
X_data = data.loc[:, data.columns.str.contains('feature')]
y_data = np.stack([(data[c] > 0).astype('int') for c in resp_cols]).T

hidden_units = [300, 200, 50] # 之前调整过的网络结构

# 二分类阈值
threshold = 0.5
# 对验证集的预测
y_oof = np.zeros(data.shape[0])
# 避免第一折训练集太小，只切分了3折
```

```
tkf = TimeSeriesSplit(n_splits=3)
for train_index, valid_index in tkf.split(X_data):
    X_train, X_valid = X_data[train_index], X_data[valid_index]
    y_train, y_valid = y_data[train_index], y_data[valid_index]

    inp = tf.keras.layers.Input(shape=(130,))
    x = tf.keras.layers.Dropout(0.2)(inp)
    for i in range(len(hidden_units)):
        x = tf.keras.layers.Dense(units=hidden_units[i],activation='relu')
(x)
        x = tf.keras.layers.Dropout(0.2)(x)
    x = tf.keras.layers.Dense(5)(x)
    out = tf.keras.layers.Activation("sigmoid")(x)
    model = tf.keras.models.Model(inputs=inp, outputs=out)
    model.compile(
        optimizer=tf.keras.optimizers.Adam(0.001),
        loss=['binary_crossentropy'],
        metrics=tf.keras.metrics.AUC(name="auc")
    )

    model.fit(X_train,y_train,validation_data=
(X_valid,y_valid),epochs=200,batch_size=5000,callbacks=
[EarlyStopping('val_auc',mode='max',patience=15,restore_best_weights=True)])

    pred = model.predict(X_valid)
    pred = np.median(pred, axis=1) # 预测的五标签取中位数
    pred = np.where(pred >= threshold, 1, 0).astype(int)
    y_oof[valid_index] = pred
```

提交结果，线上得分为7253。

5. 进阶模型

Baseline分数较低，于是参考了Kaggle论坛中的一个开源模型。该模型由两部分组成：

1. 降噪自编码器 [1]

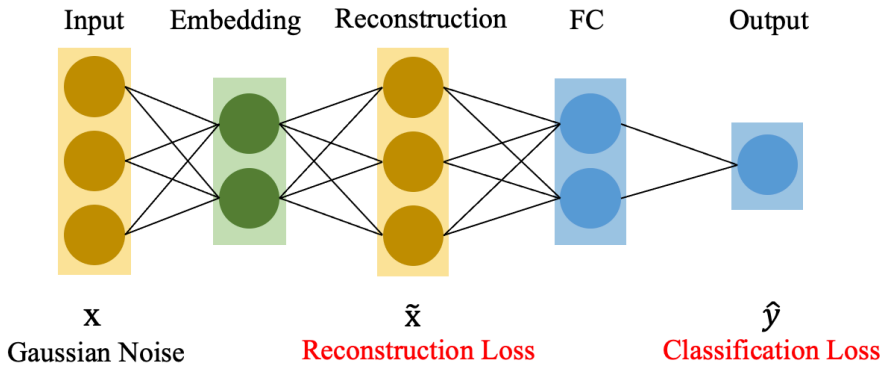


图 10

首先在样本中加入高斯噪声，传入自编码器重建原始特征，计算**重建损失**；之后将重建特征传入由全连接神经网络构成的分类器，计算**分类损失**，利用这两个损失来训练网络。

使用全部数据对模型进行训练，之后将训练好的编码器部分，加入到第二个模型中。

自编码器的作用是降维，但是容易过拟合，因此在自编码器的输入层中加入高斯噪声有助于提高模型的鲁棒性。此外，在训练模型时加入分类损失的监督，可以使学习到的表征反映数据的类别信息。

```
def create_autoencoder(input_dim,output_dim,noise=0.05):
    i = Input(input_dim)
    encoded = BatchNormalization()(i)
    encoded = GaussianNoise(noise)(encoded)
    encoded = Dense(64,activation='relu')(encoded)
    decoded = Dropout(0.2)(encoded)
    decoded = Dense(input_dim,name='decoded')(decoded)
    x = Dense(32,activation='relu')(decoded)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(32,activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    x = Dense(output_dim,activation='sigmoid',name='label_output')(x)

    encoder = Model(inputs=i,outputs=encoded)
    autoencoder = Model(inputs=i,outputs=[decoded,x])

    autoencoder.compile(optimizer=Adam(0.005),loss=
{'decoded':'mse','label_output':'binary_crossentropy'})
    return autoencoder, encoder

autoencoder, encoder = create_autoencoder(X.shape[-1],y.shape[-1],noise=0.1)
autoencoder.fit(X,(X,y),
                    epochs=1000,
                    batch_size=4096,
                    validation_split=0.1,
                    callbacks=
[EarlyStopping('val_loss',patience=10,restore_best_weights=True)])
```

2. 全连接神经网络分类器

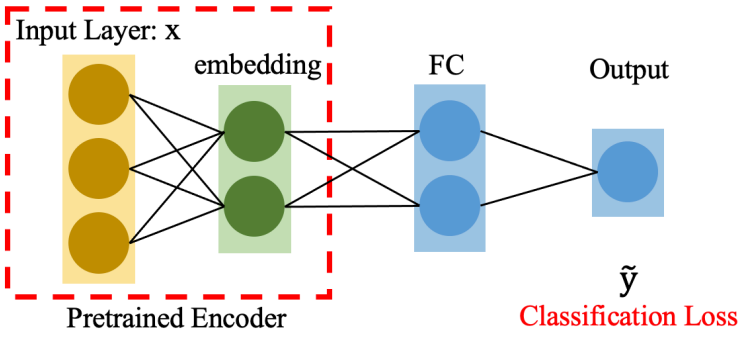


图 11

将训练好的编码器作为分类器的输入部分，经过全连接层做出分类预测，使用得到的**分类损失**训练整个模型。

```
def create_model(input_dim,output_dim,encoder):
    inputs = Input(input_dim)

    x = encoder(inputs)
    x = Concatenate()([x,inputs]) #use both raw and encoded features
    x = BatchNormalization()(x)
    x = Dropout(0.13)(x)

    hidden_units = [384, 896, 896, 394]
    for idx, hidden_unit in enumerate(hidden_units):
        x = Dense(hidden_unit)(x)
        x = BatchNormalization()(x)
        x = Lambda(tf.keras.activations.relu)(x)
        x = Dropout(0.25)(x)
    x = Dense(output_dim,activation='sigmoid')(x)
    model = Model(inputs=inputs,outputs=x)

model.compile(optimizer=Adam(0.0005),loss=BinaryCrossentropy(label_smoothing
=0.05),metrics=[tf.keras.metrics.AUC(name = 'auc')])
    return model

// 训练模型
for train_index, valid_index in tkf.split(X_data):
    X_train, X_valid = X_data[train_index], X_data[valid_index]
    y_train, y_valid = y_data[train_index], y_data[valid_index]

    model = create_model(130, 5, encoder)
    model.fit(X_train,y_train,validation_data=
(X_test,y_test),epochs=200,batch_size=4096,callbacks=
[EarlyStopping('val_auc',mode='max',patience=10,restore_best_weights=True)])
```

提交结果，线上得分为8766。

6. 树模型

6.1 特征工程

基于训练数据构建XGBoost模型，查看树模型给出的特征重要性排序，对重要特征进行特征组合。

```
# 只使用resp标签
y = (data['resp'] > 0).astype('int').T
train_X = X_data[:1000000,:]
train_y = y[:1000000]
valid_X = X_data[1000000:1200000,:]
valid_y = y[1000000:1200000]

train_data = xgb.DMatrix(train_X, train_y)
valid_data = xgb.DMatrix(valid_X, valid_y)

watchlist = [(valid_data, 'validation_auc')]
# 基本使用默认参数
xgb_params = {
    'booster':'gbtree',
    'objective':'binary:logistic',
    'eta': 0.05,
    'eval_metric': 'auc',
    'tree_method': 'gpu_hist'
}

xgb_clf = xgb.train(
                    xgb_params,
                    train_data,
                    num_boost_round=1000,
                    early_stopping_rounds=30,
                    evals=watchlist,
                    verbose_eval=10
                    )
d = xgb_clf.get_fscore()
xgb.plot_importance(xgb_clf,max_num_features=20)
```

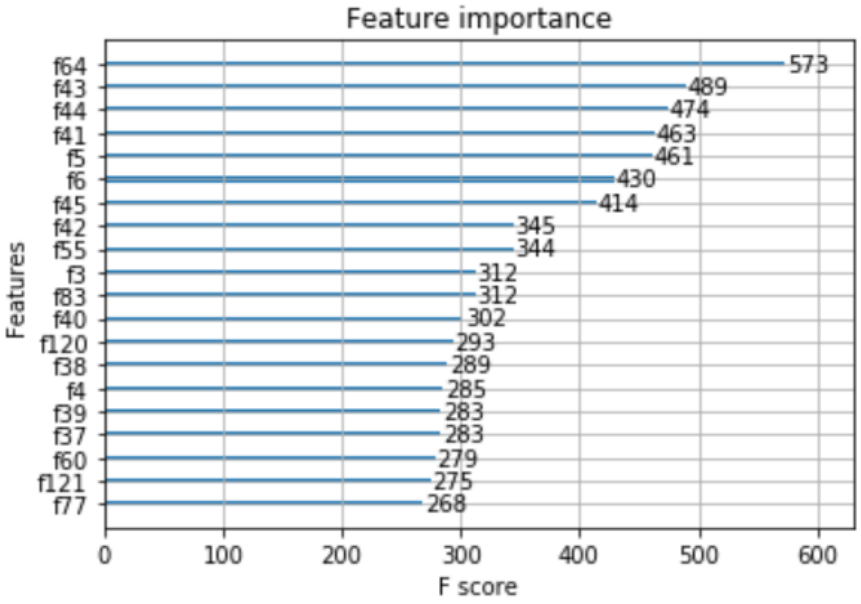


图 12

我们选取了最重要的20个特征，两两组合 (相乘 & 相加)，生成了420个新的特征。

```
temp_data = data
for i in range(20):
    for j in range(i,20):
        f1 = important_f_list[i]
        f2 = important_f_list[j]
        temp_data[f1+'*'+f2] = temp_data[f1] * temp_data[f2]
        temp_data[f1+'+'+f2] = temp_data[f1] + temp_data[f2]
```

新生成的特征很多， 因此要对一部分冗余的特征进行删除。使用的方法是删除相关性高（相关度>0.9）的新特征。

```
X_data = temp_data.loc[:, temp_data.columns.str.contains('feature')]
corr = X_data.corr(method='spearman')
c_l = corr.columns.values.tolist()
redunant_list = set()
for i in range(len(c_l)):
    for j in range(i+1,len(c_l)):
        f1 = c_l[i]
        f2 = c_l[j]
        if corr.loc[f1, f2] > 0.8 or corr.loc[f1, f2] < -0.8:
            redunant_list.add(f1)
```

剔除掉相关性大的特征 (新生成特征之间的相关性 & 新生成特征与原始特征之间的相关性)，最终保留了102个特征。

特征工程完毕，目前数据集的特征共232个（原始特征130个，新特征102个）。

6.2 建立模型

由于XGBoost官方没有提供多标签分类的接口，因此我们构建了五个二分类的XGBoost模型，分别预测五个标签。以下以预测resp标签的模型为例：

```
y1 = (data['resp_1']>0).astype('int').T

threshold = 0.5
y1_oof = np.zeros(data.shape[0])
tkf = TimeSeriesSplit(n_splits=3)
for train_index, valid_index in tkf.split(X_data):
    X_train, X_valid = X_data[train_index], X_data[valid_index]
    y_train, y_valid = y1[train_index], y_data[valid_index]

    train_data = xgb.DMatrix(train_X, train_y)
    valid_data = xgb.DMatrix(valid_X, valid_y)

    watchlist = [(valid_data, 'validation_auc')]

    xgb_params = {
        'booster':'gbtree',
        'objective':'binary:logistic',
        'eta': 0.05,
        'eval_metric': 'auc',
        'max_depth': 9,
        'subsample': 0.9,
        'colsample_bytree': 0.7,
        'gamma': 0.4,
        'tree_method': 'gpu_hist'
    }

    xgb_clf_1 = xgb.train(
                    xgb_params,
                    train_data,
                    num_boost_round=1000,
                    early_stopping_rounds=50,
                    evals=watchlist,
                    verbose_eval=10
                    )
    y1_oof[valid_index] = xgb_clf.predict(valid_data)
```

将提交到线上结果为8012.

7. 模型融合 & 结果提交

由于两种模型线上结果相近，因此赋予相同的权重。首先使用两种模型预测测试集的五个标签，之后对五个标签的预测概率求平均并利用阈值得到0/1标签，最终使用投票法确定对action的预测结果。

```
import janestreet

env = janestreet.make_env()
iter_test = env.iter_test()

threshold = 0.503
# threshold = 0.5
nn_w = round(8766/(8012+8766),3)
xgb_w = round(8012/(8012+8766),3)

# Kaggle Kernel赛要求的数据读取格式
for (test_df, pred_df) in tqdm(iter_test):
    if test_df['weight'].item() > 0:
        x_tt = test_df.loc[:, features].values

        if np.isnan(x_tt[:, 1:].sum()):
            # 使用训练集中的特征均值填补测试样本中的缺失
            x_tt[:, 1:] = np.nan_to_num(x_tt[:, 1:]) + np.isnan(x_tt[:, 1:]) * f_mean

        nn_pred = nn_clf.predict(x_tt)

        test_data = xgb.DMatrix(x_tt)
        xgb_pred_1 = xgb_clf_1.predict(test_data)
        xgb_pred_2 = xgb_clf_2.predict(test_data)
        xgb_pred_3 = xgb_clf_3.predict(test_data)
        xgb_pred_4 = xgb_clf_4.predict(test_data)
        xgb_pred_0 = xgb_clf_0.predict(test_data)
        xgb_pred = np.vstack((xgb_pred_1,xgb_pred_2,xgb_pred_3,xgb_pred_4,xgb_pred_0)).T

        merge_pred = nn_pred*nn_w + xgb_pred*xgb_w
        pred = np.median(merge_pred)
        pred_df.action = np.where(pred >= threshold, 1, 0).astype(int)

    else:
        pred_df.action = 0
    env.predict(pred_df)
```

最终线上结果为9539分。

8. 未来工作

提交截止日时，成绩位于公开榜单铜牌区上游，私榜测试集切换后，排名有所下滑，证明发生了对公榜的过拟合。可以改进的地方如下：

1. 模型中存在很多专门为了拟合公榜数据的操作，例如专门调试分类阈值为threshold=0.503
2. 在为树模型做特征工程时，只使用了特征之间的交叉 (相乘&相加)。可以从时间序列的角度进行特征挖掘。

9. 参考

[1] Parviainen, Elina. "Deep bottleneck classifiers in supervised dimension reduction." International Conference on Artificial Neural Networks. Springer, Berlin, Heidelberg, 2010.