CHAPTER 8
Feature Importance

8.1 MOTIVATION

One of the most pervasive mistakes in financial research is to take some data, run it through an ML algorithm, backtest the predictions, and repeat the sequence until a nice-looking backtest shows up. Academic journals are filled with such pseudo-discoveries, and even large hedge funds constantly fall into this trap. It does not matter if the backtest is a walk-forward out-of-sample. The fact that we are repeating a test over and over on the same data will likely lead to a false discovery. This methodological error is so notorious among statisticians that they consider it scientific fraud, and the American Statistical Association warns against it in its ethical guidelines (American Statistical Association [2016], Discussion #4). It typically takes about 20 such iterations to discover a (false) investment strategy subject to the standard significance level (false positive rate) of 5%. In this chapter we will explore why such an approach is a waste of time and money, and how feature importance offers an alternative.

8.2 THE IMPORTANCE OF FEATURE IMPORTANCE

A striking facet of the financial industry is that so many very seasoned portfolio managers (including many with a quantitative background) do not realize how easy it is to overfit a backtest. How to backtest properly is not the subject of this chapter; we will address that extremely important topic in Chapters 11–15. The goal of this chapter is to explain one of the analyses that must be performed *before* any backtest is carried out.

Suppose that you are given a pair of matrices (X, y), that respectively contain features and labels for a particular financial instrument. We can fit a classifier on (X, y) and evaluate the generalization error through a purged k-fold cross-validation (CV), as we saw in Chapter 7. Suppose that we achieve good performance. The next natural question is to try to understand what features contributed to that performance. Maybe we could add some features that strengthen the signal responsible for the classifier's predictive power. Maybe we

could eliminate some of the features that are only adding noise to the system. Notably, understanding feature importance opens up the proverbial black box. We can gain insight into the patterns identified by the classifier if we understand what source of information is indispensable to it. This is one of the reasons why the black box mantra is somewhat overplayed by the ML skeptics. Yes, the algorithm has learned without us directing the process (that is the whole point of ML!) in a black box, but that does not mean that we cannot (or should not) take a look at what the algorithm has found. Hunters do not blindly eat everything their smart dogs retrieve for them, do they?

Once we have found what features are important, we can learn more by conducting a number of experiments. Are these features important all the time, or only in some specific environments? What triggers a change in importance over time? Can those regime switches be predicted? Are those important features also relevant to other related financial instruments? Are they relevant to other asset classes? What are the most relevant features across all financial instruments? What is the subset of features with the highest rank correlation across the entire investment universe? This is a much better way of researching strategies than the foolish backtest cycle. Let me state this maxim as one of the most critical lessons I hope you learn from this book:

SNIPPET 8.1 MARCOS' FIRST LAW OF BACKTESTING—IGNORE AT YOUR OWN PERIL

"Backtesting is not a research tool. Feature importance is."

—Marcos López de Prado
Advances in Financial Machine Learning (2018)

8.3 FEATURE IMPORTANCE WITH SUBSTITUTION EFFECTS

I find it useful to distinguish between feature importance methods based on whether they are impacted by substitution effects. In this context, a substitution effect takes place when the estimated importance of one feature is reduced by the presence of other related features. Substitution effects are the ML analogue of what the statistics and econometrics literature calls "multi-collinearity." One way to address linear substitution effects is to apply PCA on the raw features, and

then perform the feature importance analysis on the orthogonal features. See Belsley et al. [1980], Goldberger [1991, pp. 245–253], and Hill et al. [2001] for further details.

8.3.1 Mean Decrease Impurity

Mean decrease impurity (MDI) is a fast, explanatory-importance (in-sample, IS) method specific to tree-based classifiers, like RF. At each node of each decision tree, the selected feature splits the subset it received in such a way that impurity is decreased. Therefore, we can derive for each decision tree how much of the overall impurity decrease can be assigned to each feature. And given that we have a forest of trees, we can average those values across all estimators and rank the features accordingly. See Louppe et al. [2013] for a detailed description. There are some important considerations you must keep in mind when working with MDI:

1. Masking effects take place when some features are systematically ignored by tree-based classifiers in favor of others. In order to avoid them, set `max_features=int(1)`when using sklearn's RF class. In this way, only one random feature is considered per level.
    a. Every feature is given a chance (at some random levels of some random trees) to reduce impurity.
    b. Make sure that features with zero importance are not averaged, since the only reason for a 0 is that the feature was not randomly chosen. Replace those values with `np.nan`.
2. The procedure is obviously IS. Every feature will have some importance, even if they have no predictive power whatsoever.
3. MDI cannot be generalized to other non-tree based classifiers.
4. By construction, MDI has the nice property that feature importances add up to 1, and every feature importance is bounded between 0 and 1.
5. The method does not address substitution effects in the presence of codependent features. MDI dilutes the importance of substitute features, because of their interchangeability: The importance of two identical features will be halved, as they are randomly chosen with equal probability.
6. Strobl et al. [2007] show experimentally that MDI is biased towards some predictor variables. White and Liu [1994] argue that, in case of single decision trees, this bias is due to an unfair advantage given by popular

impurity functions toward predictors with a large number of categories.

Sklearn's `RandomForest` class implements MDI as the default feature importance score. This choice is likely motivated by the ability to compute MDI on the fly, with minimum computational cost.[1] Snippet 8.2 illustrates an implementation of MDI, incorporating the considerations listed earlier.

SNIPPET 8.2 MDI FEATURE IMPORTANCE

```
def featImpMDI(fit,featNames):
    # feat importance based on IS mean impurity reduction
    df0={i:tree.feature_importances_ for i,tree in enumerate(fit.estimators_)}
    df0=pd.DataFrame.from_dict(df0,orient='index')
    df0.columns=featNames
    df0=df0.replace(0,np.nan) # because max_features=1
    imp=pd.concat({'mean':df0.mean(),'std':df0.std()*df0.shape[0]**-.5},axis=1)
    imp/=imp['mean'].sum()
    return imp
```

8.3.2 Mean Decrease Accuracy

Mean decrease accuracy (MDA) is a slow, predictive-importance (out-of-sample, OOS) method. First, it fits a classifier; second, it derives its performance OOS according to some performance score (accuracy, negative log-loss, etc.); third, it permutates (i.e., shuffles) each column of the features matrix (X), one column at a time, deriving the performance OOS after each column's permutation. The importance of a feature is a function of the loss in performance caused by its column's permutation. Some relevant considerations include:

1. This method can be applied to any classifier, not only tree-based classifiers.
2. MDA is not limited to accuracy as the sole performance score. For example, in the context of meta-labeling applications, we may prefer to score a classifier with F1 rather than accuracy (see Chapter 14, Section 14.8 for an explanation). That is one reason a better descriptive name would have been "permutation importance." When the scoring function does not correspond to a metric space, MDA results should be used as a ranking.
3. Like MDI, the procedure is also susceptible to substitution effects in the presence of codependent features. Given two identical features, shuffling one still allows the other one to contribute to the prediction, hence giving the false impression that both features are less important than they truly are.

4. Unlike MDI, it is possible that MDA concludes that all features are unimportant. That is because MDA is based on OOS performance.
5. The CV must be purged and embargoed, for the reasons explained in Chapter 7.

Snippet 8.3 implements MDA feature importance with sample weights, with purged k-fold CV, and with scoring by negative log-loss or accuracy. It measures MDA importance as a function of the improvement (from permutating to not permutating the feature), relative to the maximum possible score (negative log-loss of 0, or accuracy of 1). Note that, in some cases, the improvement may be negative, meaning that the feature is actually detrimental to the forecasting power of the ML algorithm.

SNIPPET 8.3 MDA FEATURE IMPORTANCE

```
def featImpMDA(clf,X,y,cv,sample_weight,t1,pctEmbargo,scoring='neg_log_loss'):
    # feat importance based on OOS score reduction
    if scoring not in ['neg_log_loss','accuracy']:
        raise Exception('wrong scoring method.')
    from sklearn.metrics import log_loss,accuracy_score
    cvGen=PurgedKFold(n_splits=cv,t1=t1,pctEmbargo=pctEmbargo) # purged cv
    scr0,scr1=pd.Series(),pd.DataFrame(columns=X.columns)
    for i,(train,test) in enumerate(cvGen.split(X=X)):
        X0,y0,w0=X.iloc[train,:],y.iloc[train],sample_weight.iloc[train]
        X1,y1,w1=X.iloc[test,:],y.iloc[test],sample_weight.iloc[test]
        fit=clf.fit(X=X0,y=y0,sample_weight=w0.values)
        if scoring=='neg_log_loss':
            prob=fit.predict_proba(X1)
            scr0.loc[i]=-log_loss(y1,prob,sample_weight=w1.values,
                                  labels=clf.classes_)
        else:
            pred=fit.predict(X1)
            scr0.loc[i]=accuracy_score(y1,pred,sample_weight=w1.values)
        for j in X.columns:
            X1_=X1.copy(deep=True)
            np.random.shuffle(X1_[j].values) # permutation of a single column
            if scoring=='neg_log_loss':
                prob=fit.predict_proba(X1_)
                scr1.loc[i,j]=-log_loss(y1,prob,sample_weight=w1.values,
                                        labels=clf.classes_)
            else:
                pred=fit.predict(X1_)
                scr1.loc[i,j]=accuracy_score(y1,pred,sample_weight=w1.values)
    imp=(-scr1).add(scr0,axis=0)
    if scoring=='neg_log_loss':imp=imp/-scr1
    else:imp=imp/(1.-scr1)
    imp=pd.concat({'mean':imp.mean(),'std':imp.std()*imp.shape[0]**-.5},axis=1)
    return imp,scr0.mean()
```

8.4 FEATURE IMPORTANCE WITHOUT SUBSTITUTION EFFECTS

Substitution effects can lead us to discard important features that happen to be redundant. This is not generally a problem in the context of prediction, but it could lead us to wrong conclusions when we are trying to understand, improve, or simplify a model. For this reason, the following single feature importance method can be a good complement to MDI and MDA.

8.4.1 Single Feature Importance

Single feature importance (SFI) is a cross-section predictive-importance (out-of-sample) method. It computes the OOS performance score of each feature in isolation. A few considerations:

1. This method can be applied to any classifier, not only tree-based classifiers.
2. SFI is not limited to accuracy as the sole performance score.
3. Unlike MDI and MDA, no substitution effects take place, since only one feature is taken into consideration at a time.
4. Like MDA, it can conclude that all features are unimportant, because performance is evaluated via OOS CV.

The main limitation of SFI is that a classifier with two features can perform better than the bagging of two single-feature classifiers. For example, (1) feature B may be useful only in combination with feature A; or (2) feature B may be useful in explaining the splits from feature A, even if feature B alone is inaccurate. In other words, joint effects and hierarchical importance are lost in SFI. One alternative would be to compute the OOS performance score from subsets of features, but that calculation will become intractable as more features are considered. Snippet 8.4 demonstrates one possible implementation of the SFI method. A discussion of the function `cvScore` can be found in Chapter 7.

SNIPPET 8.4 IMPLEMENTATION OF SFI

```
def auxFeatImpSFI(featNames,clf,trnsX,cont,scoring,cvGen):
    imp=pd.DataFrame(columns=['mean','std'])
    for featName in featNames:
        df0=cvScore(clf,X=trnsX[[featName]],y=cont['bin'],sample_weight=cont['w'],
                    scoring=scoring,cvGen=cvGen)
        imp.loc[featName,'mean']=df0.mean()
        imp.loc[featName,'std']=df0.std()*df0.shape[0]**-.5
    return imp
```

8.4.2 Orthogonal Features

As argued in Section 8.3, substitution effects dilute the importance of features measured by MDI, and significantly underestimate the importance of features measured by MDA. A partial solution is to orthogonalize the features before applying MDI and MDA. An orthogonalization procedure such as principal components analysis (PCA) does not prevent all substitution effects, but at least it should alleviate the impact of linear substitution effects.

Consider a matrix $\{X_{t,n}\}$ of stationary features, with observations $t = 1,\ldots, T$ and variables $n = 1,\ldots, N$. First, we compute the standardized features matrix Z, such that $Z_{t,n} = \sigma_n^{-1}(X_{t,n} - \mu_n)$, where μ_n is the mean of $\{X_{t,n}\}_{t = 1,\ldots, T}$ and σ_n is the standard deviation of $\{X_{t,n}\}_{t = 1,\ldots, T}$. Second, we compute the eigenvalues Λ and eigenvectors W such that $Z'ZW = W\Lambda$, where Λ is an NxN diagonal matrix with main entries sorted in descending order, and W is an NxN orthonormal matrix. Third, we derive the orthogonal features as $P = ZW$. We can verify the orthogonality of the features by noting that $P'P = W'Z'ZW = W'W\Lambda W'W = \Lambda$.

The diagonalization is done on Z rather than X, for two reasons: (1) centering the data ensures that the first principal component is correctly oriented in the main direction of the observations. It is equivalent to adding an intercept in a linear regression; (2) re-scaling the data makes PCA focus on explaining correlations rather than variances. Without re-scaling, the first principal components would be dominated by the columns of X with highest variance, and we would not learn much about the structure or relationship between the variables.

Snippet 8.5 computes the smallest number of orthogonal features that explain at least 95% of the variance of Z.

SNIPPET 8.5 COMPUTATION OF ORTHOGONAL FEATURES

```
def get_eVec(dot,varThres):
    # compute eVec from dot prod matrix, reduce dimension
    eVal,eVec=np.linalg.eigh(dot)
    idx=eVal.argsort()[::-1] # arguments for sorting eVal desc
    eVal,eVec=eVal[idx],eVec[:,idx]
    #2) only positive eVals
    eVal=pd.Series(eVal,index=['PC_'+str(i+1) for i in range(eVal.shape[0])])
    eVec=pd.DataFrame(eVec,index=dot.index,columns=eVal.index)
    eVec=eVec.loc[:,eVal.index]
    #3) reduce dimension, form PCs
    cumVar=eVal.cumsum()/eVal.sum()
    dim=cumVar.values.searchsorted(varThres)
    eVal,eVec=eVal.iloc[:dim+1],eVec.iloc[:,:dim+1]
    return eVal,eVec
#---------------------------------------------------------------
def orthoFeats(dfX,varThres=.95):
    # Given a dataframe dfX of features, compute orthofeatures dfP
    dfZ=dfX.sub(dfX.mean(),axis=1).div(dfX.std(),axis=1) # standardize
    dot=pd.DataFrame(np.dot(dfZ.T,dfZ),index=dfX.columns,columns=dfX.columns)
    eVal,eVec=get_eVec(dot,varThres)
    dfP=np.dot(dfZ,eVec)
    return dfP
```

Besides addressing substitution effects, working with orthogonal features provides two additional benefits: (1) orthogonalization can also be used to reduce the dimensionality of the features matrix X, by dropping features associated with small eigenvalues. This usually speeds up the convergence of ML algorithms; (2) the analysis is conducted on features designed to explain the structure of the data.

Let me stress this latter point. An ubiquitous concern throughout the book is the risk of overfitting. ML algorithms will always find a pattern, even if that pattern is a statistical fluke. You should always be skeptical about the purportedly important features identified by any method, including MDI, MDA, and SFI. Now, suppose that you derive orthogonal features using PCA. Your PCA analysis has determined that some features are more "principal" than others, without any knowledge of the labels (unsupervised learning). That is, PCA has ranked features without any possible overfitting in a classification sense. When your MDI, MDA, or SFI analysis selects as most important (using label information) the same features that PCA chose as principal (ignoring label information), this constitutes confirmatory evidence that the pattern identified by the ML algorithm is not entirely overfit. If the features were entirely random, the PCA ranking would have no correspondance with the feature importance ranking. Figure 8.1 displays the scatter plot of eigenvalues associated with an eigenvector (x-axis) paired with MDI of the feature associated with an engenvector (y-axis). The Pearson correlation is 0.8491 (p-value below 1E-150), evidencing that PCA identified informative features and ranked them correctly without overfitting.

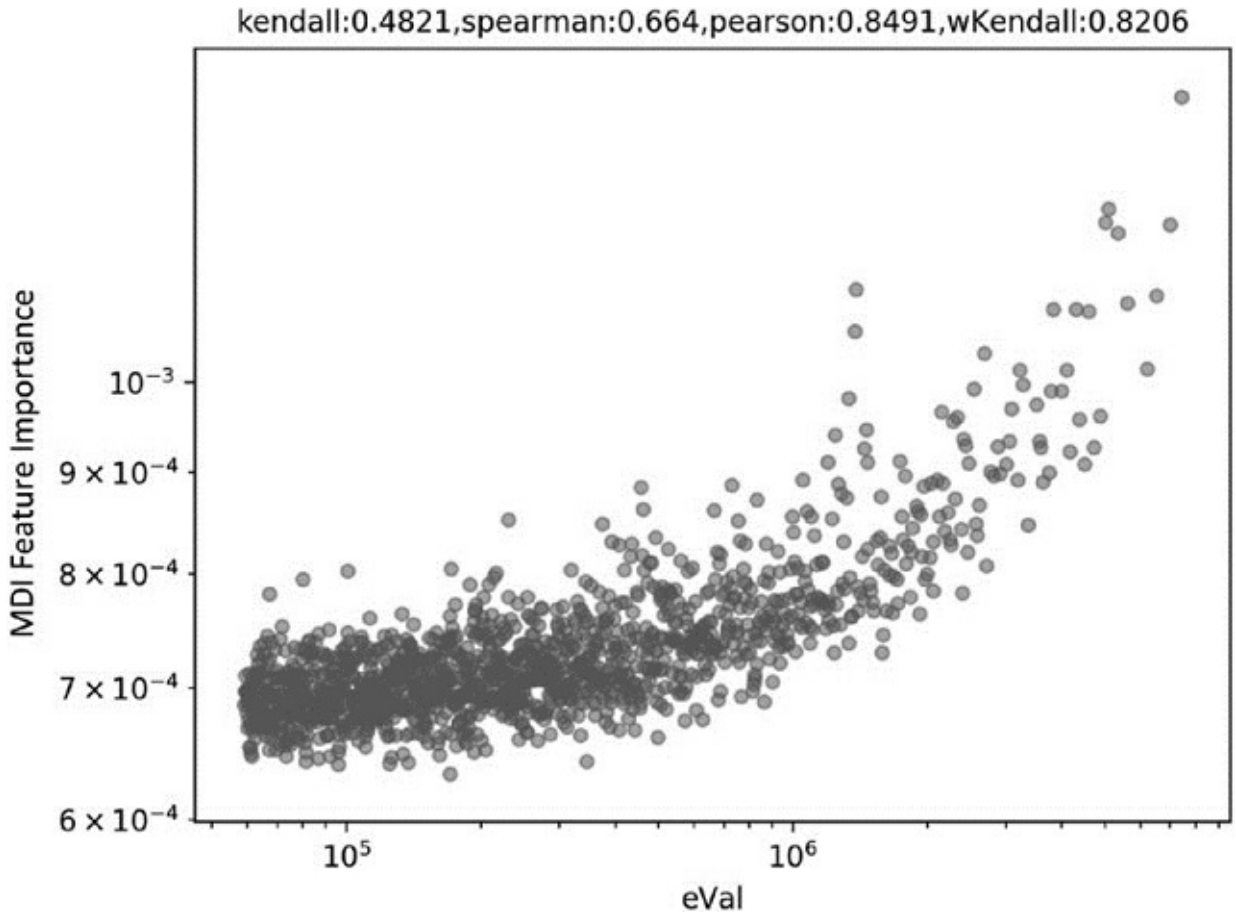

FIGURE 8.1 Scatter plot of eigenvalues (x-axis) and MDI levels (y-axis) in log-log scale

I find it useful to compute the weighted Kendall's tau between the feature importances and their associated eigenvalues (or equivalently, their inverse PCA rank). The closer this value is to 1, the stronger is the consistency between PCA ranking and feature importance ranking. One argument for preferring a weighted Kendall's tau over the standard Kendall is that we want to prioritize rank concordance among the most importance features. We do not care so much about rank concordance among irrelevant (likely noisy) features. The hyperbolic-weighted Kendall's tau for the sample in Figure 8.1 is 0.8206.

Snippet 8.6 shows how to compute this correlation using Scipy. In this example, sorting the features in descending importance gives us a PCA rank sequence very close to an ascending list. Because the `weightedtau` function gives higher weight to higher values, we compute the correlation on the inverse PCA ranking, `pcRank**-1`. The resulting weighted Kendall's tau is relatively high, at 0.8133.

SNIPPET 8.6 COMPUTATION OF WEIGHTED KENDALL'S TAU BETWEEN FEATURE IMPORTANCE AND INVERSE PCA RANKING

```
>>> import numpy as np
>>> from scipy.stats import weightedtau
>>> featImp=np.array([.55,.33,.07,.05]) # feature importance
>>> pcRank=np.array([1,2,4,3]) # PCA rank
>>> weightedtau(featImp,pcRank**-1.)[0]
```

8.5 PARALLELIZED VS. STACKED FEATURE IMPORTANCE

There are at least two research approaches to feature importance. First, for each security i in an investment universe $i = 1,\ldots, I$, we form a dataset (X_i, y_i), and derive the feature importance in parallel. For example, let us denote $\lambda_{i,j,k}$ the importance of feature j on instrument i according to criterion k. Then we can aggregate all results across the entire universe to derive a combined $\Lambda_{j,k}$ importance of feature j according to criterion k. Features that are important across a wide variety of instruments are more likely to be associated with an underlying phenomenon, particularly when these feature importances exhibit high rank correlation across the criteria. It may be worth studying in-depth the theoretical mechanism that makes these features predictive. The main advantage of this approach is that it is computationally fast, as it can be parallelized. A disadvantage is that, due to substitution effects, important features may swap their ranks across instruments, increasing the variance of the estimated $\lambda_{i,j,k}$. This disadvantage becomes relatively minor if we average $\lambda_{i,j,k}$ across instruments for a sufficiently large investment universe.

A second alternative is what I call "features stacking." It consists in stacking all datasets $\{(\tilde{X}_i, y_i)\}_{i=1,\ldots,I}$ into a single combined dataset (X, y), where $\tilde{X}_i$ is a transformed instance of X_i (e.g., standardized on a rolling trailing window). The purpose of this transformation is to ensure some distributional homogeneity, $\tilde{X}_i \sim X$. Under this approach, the classifier must learn what features are more important across all instruments simultaneously, as if the entire investment universe were in fact a single instrument. Features stacking presents some

advantages: (1) The classifier will be fit on a much larger dataset than the one used with the parallelized (first) approach; (2) the importance is derived directly, and no weighting scheme is required for combining the results; (3) conclusions are more general and less biased by outliers or overfitting; and (4) because importance scores are not averaged across instruments, substitution effects do not cause the dampening of those scores.

I usually prefer features stacking, not only for features importance but whenever a classifier can be fit on a set of instruments, including for the purpose of model prediction. That reduces the likelihood of overfitting an estimator to a particular instrument or small dataset. The main disadvantage of stacking is that it may consume a lot of memory and resources, however that is where a sound knowledge of HPC techniques will come in handy (Chapters 20–22).

8.6 EXPERIMENTS WITH SYNTHETIC DATA

In this section, we are going to test how these feature importance methods respond to synthetic data. We are going to generate a dataset (X, y) composed on three kinds of features:

1. Informative: These are features that are used to determine the label.
2. Redundant: These are random linear combinations of the informative features. They will cause substitution effects.
3. Noise: These are features that have no bearing on determining the observation's label.

Snippet 8.7 shows how we can generate a synthetic dataset of 40 features where 10 are informative, 10 are redundant, and 20 are noise, on 10,000 observations. For details on how sklearn generates synthetic datasets, visit: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html.

SNIPPET 8.7 CREATING A SYNTHETIC DATASET

```
def getTestData(n_features=40,n_informative=10,n_redundant=10,n_samples=10000):
    # generate a random dataset for a classification problem
    from sklearn.datasets import make_classification
    trnsX,cont=make_classification(n_samples=n_samples,n_features=n_features,
        n_informative=n_informative,n_redundant=n_redundant,random_state=0,
        shuffle=False)
    df0=pd.DatetimeIndex(periods=n_samples,freq=pd.tseries.offsets.BDay(),
                         end=pd.datetime.today())
    trnsX,cont=pd.DataFrame(trnsX,index=df0),
                            pd.Series(cont,index=df0).to_frame('bin')
    df0=['I_'+str(i) for i in xrange(n_informative)]+
        ['R_'+str(i) for i in xrange(n_redundant)]
    df0+=['N_'+str(i) for i in xrange(n_features-len(df0))]
    trnsX.columns=df0
    cont['w']=1./cont.shape[0]
    cont['t1']=pd.Series(cont.index,index=cont.index)
    return trnsX,cont
```

Given that we know for certain what features belong to each class, we can evaluate whether these three feature importance methods perform as designed. Now we need a function that can carry out each analysis on the same dataset. Snippet 8.8 accomplishes that, using bagged decision trees as default classifier (Chapter 6).

SNIPPET 8.8 CALLING FEATURE IMPORTANCE FOR ANY METHOD

```
def featImportance(trnsX,cont,n_estimators=1000,cv=10,max_samples=1.,numThreads=24,
                   pctEmbargo=0,scoring='accuracy',method='SFI',minWLeaf=0.,**kargs):
    # feature importance from a random forest
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import BaggingClassifier
    from mpEngine import mpPandasObj
    n_jobs=(-1 if numThreads>1 else 1) # run 1 thread with ht_helper in dirac1
    #1) prepare classifier,cv. max_features=1, to prevent masking
    clf=DecisionTreeClassifier(criterion='entropy',max_features=1,
        class_weight='balanced',min_weight_fraction_leaf=minWLeaf)
    clf=BaggingClassifier(base_estimator=clf,n_estimators=n_estimators,
        max_features=1.,max_samples=max_samples,oob_score=True,n_jobs=n_jobs)
    fit=clf.fit(X=trnsX,y=cont['bin'],sample_weight=cont['w'].values)
    oob=fit.oob_score_
    if method=='MDI':
        imp=featImpMDI(fit,featNames=trnsX.columns)
        oos=cvScore(clf,X=trnsX,y=cont['bin'],cv=cv,sample_weight=cont['w'],
                    t1=cont['t1'],pctEmbargo=pctEmbargo,scoring=scoring).mean()
    elif method=='MDA':
        imp,oos=featImpMDA(clf,X=trnsX,y=cont['bin'],cv=cv,sample_weight=cont['w'],
                           t1=cont['t1'],pctEmbargo=pctEmbargo,scoring=scoring)
    elif method=='SFI':
        cvGen=PurgedKFold(n_splits=cv,t1=cont['t1'],pctEmbargo=pctEmbargo)
        oos=cvScore(clf,X=trnsX,y=cont['bin'],sample_weight=cont['w'],scoring=scoring,
            cvGen=cvGen).mean()
        clf.n_jobs=1 # paralellize auxFeatImpSFI rather than clf
        imp=mpPandasObj(auxFeatImpSFI,('featNames',trnsX.columns),numThreads,
            clf=clf,trnsX=trnsX,cont=cont,scoring=scoring,cvGen=cvGen)
    return imp,oob,oos
```

Finally, we need a main function to call all components, from data generation to feature importance analysis to collection and processing of output. These tasks are performed by Snippet 8.9.

SNIPPET 8.9 CALLING ALL COMPONENTS

```
def testFunc(n_features=40,n_informative=10,n_redundant=10,n_estimators=1000,
             n_samples=10000,cv=10):
    # test the performance of the feat importance functions on artificial data
    # Nr noise features = n_features-n_informative-n_redundant
    trnsX,cont=getTestData(n_features,n_informative,n_redundant,n_samples)
dict0={'minWLeaf':[0.],'scoring':['accuracy'],'method':['MDI','MDA','SFI'],
    'max_samples':[1.]}
jobs,out=(dict(izip(dict0,i)) for i in product(*dict0.values())),[]
kargs={'pathOut':'./testFunc/','n_estimators':n_estimators,
    'tag':'testFunc','cv':cv}
for job in jobs:
    job['simNum']=job['method']+'_'+job['scoring']+'_'+'%.2f'%job['minWLeaf']+ \
                  '_'+str(job['max_samples'])
    print job['simNum']
    kargs.update(job)
    imp,oob,oos=featImportance(trnsX=trnsX,cont=cont,**kargs)
    plotFeatImportance(imp=imp,oob=oob,oos=oos,**kargs)
    df0=imp[['mean']]/imp['mean'].abs().sum()
    df0['type']=[i[0] for i in df0.index]
    df0=df0.groupby('type')['mean'].sum().to_dict()
    df0.update({'oob':oob,'oos':oos});df0.update(job)
    out.append(df0)
out=pd.DataFrame(out).sort_values(['method','scoring','minWLeaf','max_samples'])
out=out['method','scoring','minWLeaf','max_samples','I','R','N','oob','oos']
out.to_csv(kargs['pathOut']+'stats.csv')
return
```

For the aesthetically inclined, Snippet 8.10 provides a nice layout for plotting feature importances.

SNIPPET 8.10 FEATURE IMPORTANCE PLOTTING FUNCTION

```
def plotFeatImportance(pathOut,imp,oob,oos,method,tag=0,simNum=0,**kargs):
    # plot mean imp bars with std
    mpl.figure(figsize=(10,imp.shape[0]/5.))
    imp=imp.sort_values('mean',ascending=True)
    ax=imp['mean'].plot(kind='barh',color='b',alpha=.25,xerr=imp['std'],
                        error_kw={'ecolor':'r'})
    if method=='MDI':
        mpl.xlim([0,imp.sum(axis=1).max()])
        mpl.axvline(1./imp.shape[0],linewidth=1,color='r',linestyle='dotted')
    ax.get_yaxis().set_visible(False)
    for i,j in zip(ax.patches,imp.index):ax.text(i.get_width()/2,
                  i.get_y()+i.get_height()/2,j,ha='center',va='center',
                  color='black')
    mpl.title('tag='+tag+' | simNum='+str(simNum)+' | oob='+str(round(oob,4))+
              ' | oos='+str(round(oos,4)))
    mpl.savefig(pathOut+'featImportance_'+str(simNum)+'.png',dpi=100)
    mpl.clf();mpl.close()
    return
```

Figure 8.2 shows results for MDI. For each feature, the horizontal bar indicates the mean MDI value across all the decision trees, and the horizontal line is the standard deviation of that mean. Since MDI importances add up to 1, if all features were equally important, each importance would have a value of 1/40. The vertical dotted line marks that 1/40 threshold, separating features whose importance exceeds what would be expected from undistinguishable features. As you can see, MDI does a very good job in terms of placing all informative and redundant features above the dotted line, with the exception of R_5, which did not make the cut by a small margin. Substitution effects cause some informative or redundant features to rank better than others, which was expected.

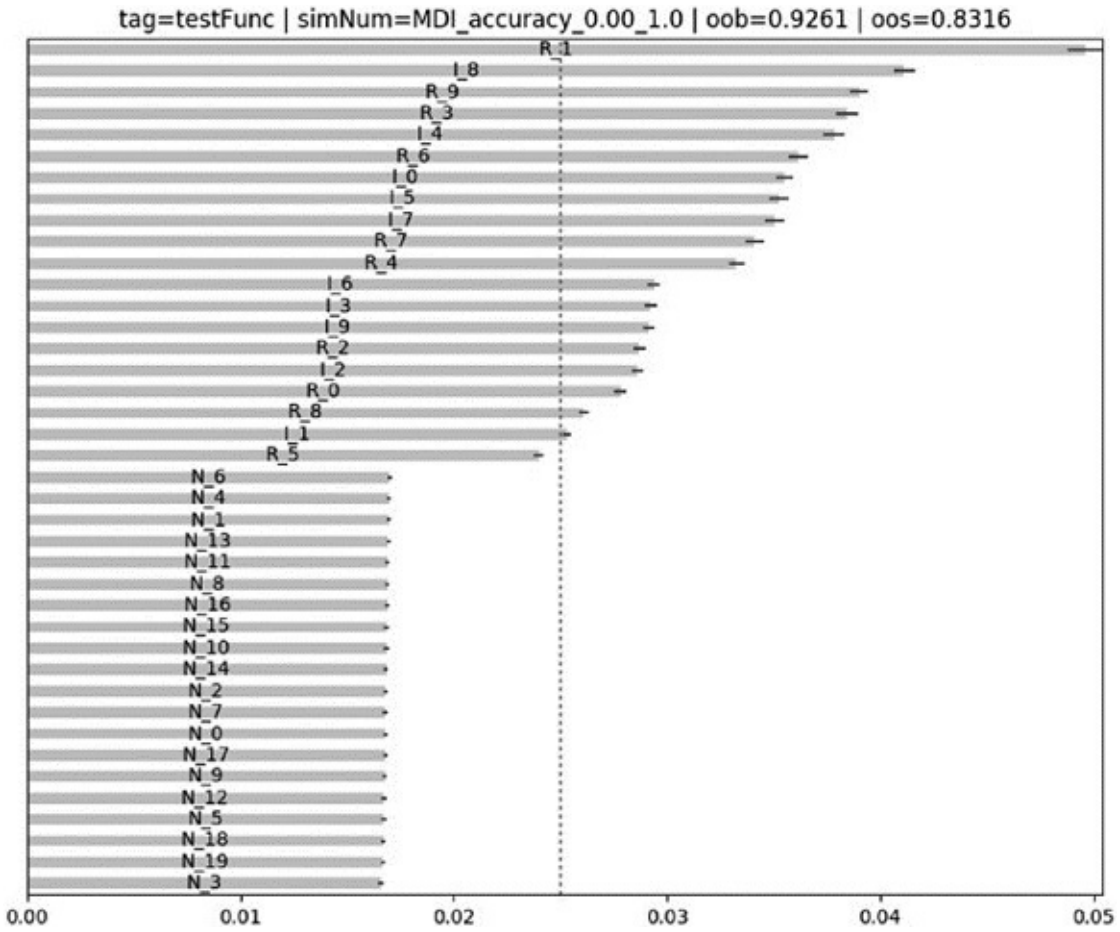

FIGURE 8.2 MDI feature importance computed on a synthetic dataset

Figure 8.3 shows that MDA also did a good job. Results are consistent with those from MDI's in the sense that all the informed and redundant features rank better than the noise feature, with the exception of R_6, likely due to a substitution effect. One not so positive aspect of MDA is that the standard deviation of the means are somewhat higher, although that could be addressed by increasing the number of partitions in the purged k-fold CV, from, say, 10 to 100 (at the cost of 10 × the computation time without parallelization).

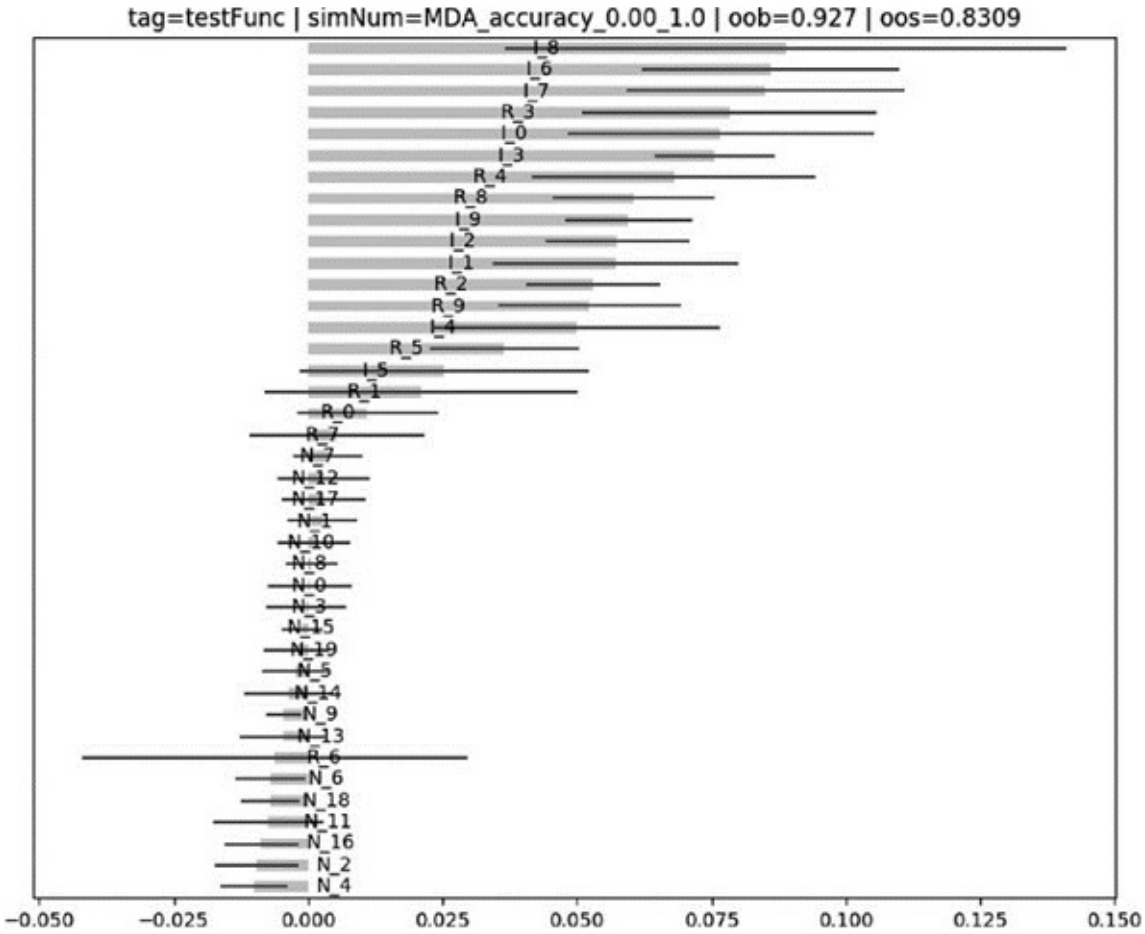

FIGURE 8.3 MDA feature importance computed on a synthetic dataset

Figure 8.4 shows that SFI also does a decent job; however, a few important features rank worse than noise (I_6, I_2, I_9, I_1, I_3, R_5), likely due to joint effects.

The labels are a function of a combination of features, and trying to forecast them independently misses the joint effects. Still, SFI is useful as a complement to MDI and MDA, precisely because both types of analyses are affected by different kinds of problems.

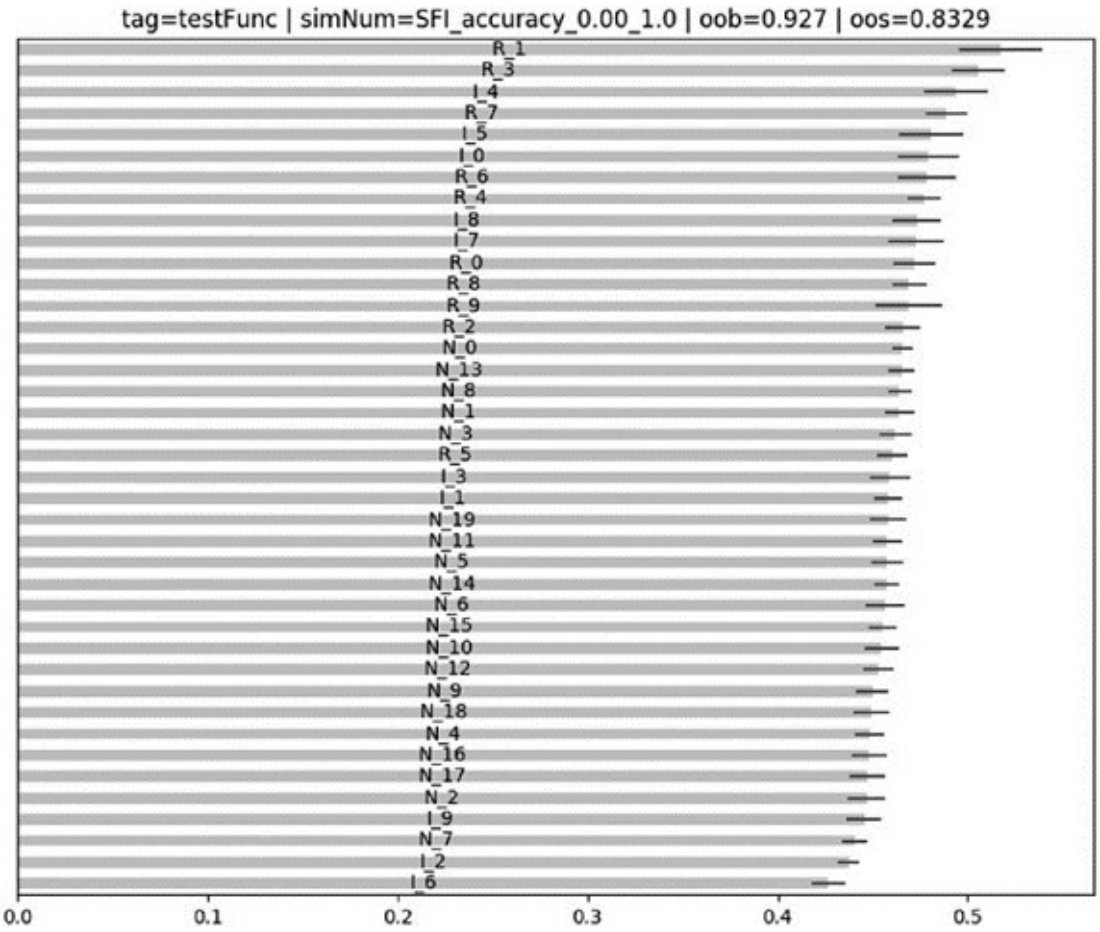

FIGURE 8.4 SFI feature importance computed on a synthetic dataset

EXERCISES

8.1 Using the code presented in Section 8.6:

a. Generate a dataset (X, y).

b. Apply a PCA transformation on X, which we denote $\dot{X}$.

c. Compute MDI, MDA, and SFI feature importance on $(\dot{X}, y)$, where the base estimator is RF.

d. Do the three methods agree on what features are important? Why?

8.2 From exercise 1, generate a new dataset $(\ddot{X}, y)$, where $\ddot{X}$ is a feature union of X and $\dot{X}$.

a. Compute MDI, MDA, and SFI feature importance on $(\ddot{X}, y)$, where the base estimator is RF.

b. Do the three methods agree on the important features? Why?

8.3 Take the results from exercise 2:

a. Drop the most important features according to each method, resulting in a features matrix $\ddot{X}$.

b. Compute MDI, MDA, and SFI feature importance on $(\ddot{X}, y)$, where the base estimator is RF.

c. Do you appreciate significant changes in the rankings of important features, relative to the results from exercise 2?

8.4 Using the code presented in Section 8.6:

a. Generate a dataset (X, y) of 1E6 observations, where 5 features are informative, 5 are redundant and 10 are noise.

b. Split (X, y) into 10 datasets $\{(X_i, y_i)\}_{i = 1,\ldots, 10}$, each of 1E5 observations.

c. Compute the parallelized feature importance (Section 8.5), on each of the 10 datasets, $\{(X_i, y_i)\}_{i = 1,\ldots, 10}$.

d. Compute the stacked feature importance on the combined dataset (X, y).

e. What causes the discrepancy between the two? Which one is more reliable?

8.5 Repeat all MDI calculations from exercises 1–4, but this time allow for masking effects. That means, do not set `max_features=int(1)` in Snippet 8.2. How do results differ as a consequence of this change? Why?

REFERENCES

American Statistical Association (2016): "Ethical guidelines for statistical practice." Committee on Professional Ethics of the American Statistical Association (April). Available at http://www.amstat.org/asa/files/pdfs/EthicalGuidelines.pdf.

Belsley, D., E. Kuh, and R. Welsch (1980): *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, 1st ed. John Wiley & Sons.

Goldberger, A. (1991): *A Course in Econometrics*. Harvard University Press, 1st edition.

Hill, R. and L. Adkins (2001): “Collinearity.” In Baltagi, Badi H. *A Companion to Theoretical Econometrics*, 1st ed. Blackwell, pp. 256–278.

Louppe, G., L. Wehenkel, A. Sutera, and P. Geurts (2013): “Understanding variable importances in forests of randomized trees.” Proceedings of the 26th International Conference on Neural Information Processing Systems, pp. 431–439.

Strobl, C., A. Boulesteix, A. Zeileis, and T. Hothorn (2007): “Bias in random forest variable importance measures: Illustrations, sources and a solution.” *BMC Bioinformatics*, Vol. 8, No. 25, pp. 1–11.

White, A. and W. Liu (1994): “Technical note: Bias in information-based measures in decision tree induction.” *Machine Learning*, Vol. 15, No. 3, pp. 321–329.

NOTE

[1] http://blog.datadive.net/selecting-good-features-part-iii-random-forests/.